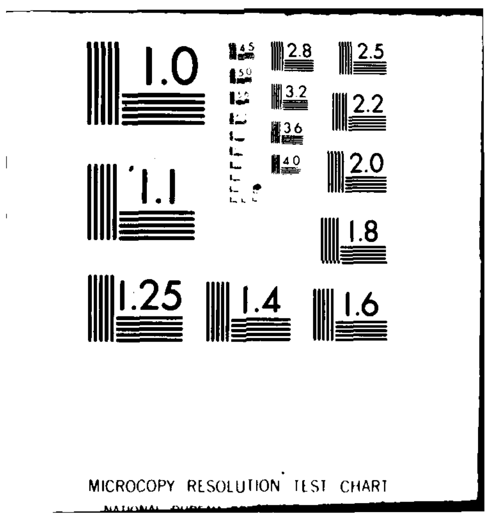
AD-A113 349 STANFORD UNIV CA DEPT OF COMPUTER SCIENCE F/G 12/1
COMPUTATION OF MATRIX CHAIN PRODUCTS. PART I, PART II.(U)
SEP 81 T C HU, M T SHING DAAG29-80-C-0029
UNCLASSIFIED STAN-CS-81-875 ARO-16323.2-M NL

1 of 2


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART

September 1981 Report. No. STAN-CS-81-875

Computation of Matrix Chain Products, Part I, Part II

by

T. C. Hu and M. T. Shing

Department of Computer Science

Stanford University
Stanford, CA 94305

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(10)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: 16323.2-M	2. GOVT ACCESSION NO.: AD-A113349
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Computation of Matrix Chain Products, Part I, Part II	5. TYPE OF REPORT & PERIOD COVERED: Technical
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): T. C. Hu; M. T. Shing	8. CONTRACT OR GRANT NUMBER(s): DAAG29 80 C 0029
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Stanford University, Stanford, CA 94305	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: U. S. Army Research Office, Post Office Box 12211, Research Triangle Park, NC 27709	12. REPORT DATE: Sep 81
	13. NUMBER OF PAGES: 99
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

DTIC ELECTE APR 12 1982 D H

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

NA

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

matrices (mathematics)
computation
chain products
algorithms

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper considers the computation of matrix chain products of the form $M_1 \times M_2 \times \ldots \times M_{n-1}$. If the matrices are of different dimensions, the order in which the product is computed affects the number of operations. An optimum order is an order which minimizes the total number of operations. We present some theorems about an optimum order of computing the matrices. Based on these theorems, and O(n log n) algorithm for finding an optimum order is presented in part II.

AD-A113349

DTIC FILE COPY

DD FORM 1473, 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Rev. Aug. 31, 1981

Computation of Matrix Chain Products, Part I

T. C. Hu and M. T. Shing

University of California, San Diego

La Jolla, CA 92093

Abstract:

This paper considers the computation of matrix chain products of the form $M_1 \times M_2 \times \cdots \times M_{n-1}$. If the matrices are of different dimensions, the order in which the product is computed affects the number of operations. An optimum order is an order which minimizes the total number of operations. We present some theorems about an optimum order of computing the matrices. Based on these theorems, an O(n log n) algorithm for finding an optimum order is presented in part II.

This research was supported in part by National Science Foundation grant MCS-77-23738 and U.S. Army Research Office grant DAAG29-80-C-0029.

1. Introduction

Consider the evaluation of the product of $n-1$ matrices

$$M = M_1 \times M_2 \times \cdots \times M_{n-1} \tag{1}$$

where M_i is a $w_i \times w_{i+1}$ matrix. Since matrix multiplication satisfies the associative law, the final result M in (1) is the same for all orders of multiplying the matrices. However, the order of multiplication greatly affects the total number of operations to evaluate M. The problem is to find an optimum order of multiplying the matrices such that the total number of operations is minimized. Here, we assume that the number of operations to multiply a $p \times q$ matrix by a $q \times r$ matrix is pqr.

In [1][7], a dynamic programming algorithm is used to find an optimum order. The algorithm needs $O(n^3)$ time and $O(n^2)$ space. In [2], Chandra proposed a heuristic algorithm to find an order of computation which requires no more than $2T_o$ operations where T_o is the total number of operations to evaluate (1) in an optimum order. This heuristic algorithm needs only $O(n)$ time. Chin [3] proposed an improved heuristic algorithm to give an order of computation which requires no more than $1.25\ T_o$. This improved heuristic algorithm also needs only $O(n)$ time.

In this paper we first transform the matrix chain product problem into a problem in graph theory - the problem of partitioning a convex polygon into non-intersecting triangles, see [9][10][11][12], then we state several theorems about the optimum partitioning problem. Based on these theorems, an $O(n \log n)$ algorithm for finding an optimum partition is developed.

2. Partitioning a convex polygon

Given an n-sided convex polygon, such as the hexagon shown in Fig. 1, the number of ways to partition the polygon into (n-2) triangles by non-intersecting diagonals is the Catalan numbers (see for example, Gould [8]). Thus, there are 2 ways to partition a convex quadrilateral, 5 ways to partition a convex pentagon, and 14 ways to partition a convex hexagon.

Let every vertex V_i of the polygon have a positive weight w_i. We can define the cost of a given partition as follows: The cost of a triangle is the product of the weights of the three vertices, and the cost of partitioning a polygon is the sum of the costs of all its triangles. For example, the cost of the partition of the hexagon in Fig. 1 is

$$w_1 w_2 w_3 + w_1 w_3 w_6 + w_3 w_4 w_6 + w_4 w_5 w_6 . \tag{2}$$

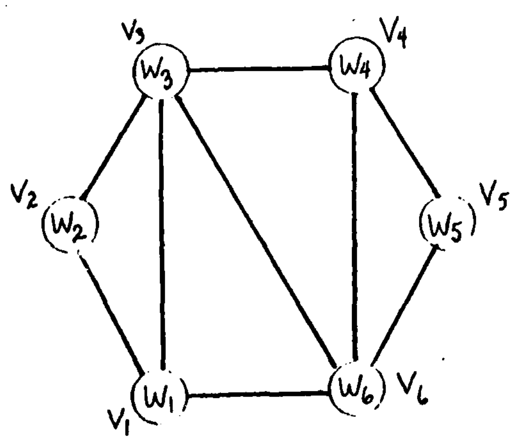


Fig. 1

If we erase the diagonal from V_3 to V_6 and replace it by the diagonal from V_1 to V_4, then the cost of the new partition will be

$$w_1 w_2 w_3 + w_1 w_3 w_4 + w_1 w_4 w_6 + w_4 w_5 w_6 \quad . \tag{3}$$

We will prove that an order of multiplying (n-1) matrices corresponds to a partition of a convex polygon with n sides. The cost of the partition is the total number of operations needed in multiplying the matrices. For brevity, we shall use n-gon to mean a convex polygon with n sides, and the partition of an n-gon to mean the partitioning of an n-gon into (n-2) non-intersecting triangles.

For any n-gon, one side of the n-gon will be considered to be its base, and will usually be drawn horizontally at the bottom such as the side V_1-V_6 in Fig. 1. This side will be called the base, all other sides are considered in a clockwise way. Thus, V_1-V_2 is the first side, V_2-V_3 the second side, ..., and V_5-V_6 the fifth side.

The first side represents the first matrix in the matrix chain and the base represents the final result M in (1). The dimensions of a matrix are the two weights associated with the two end vertices of the side. Since the adjacent matrices are compatible, the dimensions $w_1 \times w_2$, $w_2 \times w_3$, $\ldots, w_{n-1} \times w_n$ can be written inside the vertices as $w_1, w_2, \ldots, w_n$. The diagonals are the partial products. A partition of an n-gon corresponds to an alphabetic tree of n-1 leaves or the parenthesis problem of n-1 symbols (see, for example, Gardner [6]). It is easy to see the one-to-one correspondence between the multiplication of n-1 matrices to either

the alphabetic binary tree or the parenthesis problem of $n-1$ symbols. Here, we establish the correspondence between the matrix-chain product and the partition of a convex polygon directly.

Lemma 1. Any order of multiplying $n-1$ matrices corresponds to a partition of an n-gon.

Proof. We shall use induction on the number of matrices. For two matrices of dimensions $w_1 \times w_2$, $w_2 \times w_3$, there is only one way of multiplication, this corresponds to a triangle where no further partition is required. The total number of operations in multiplication is $w_1 w_2 w_3$, the product of the three weights of the vertices. The resulting matrix has dimension $w_1 \times w_3$. For three matrices, the two orders of multiplication $(M_1 \times M_2) \times M_3$ and $M_1 \times (M_2 \times M_3)$ correspond to the two ways of partitioning a 4-gon. Assume that this lemma is true for k matrices where $k \leq n-2$, and we now consider $n-1$ matrices. The n-gon is shown in Fig. 2.

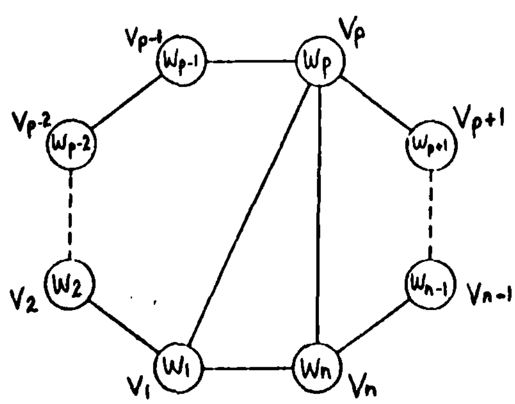


Fig. 2

Let the order of multiplication be represented by

$$M = (M_1 \times M_2 \times \cdots \times M_{p-1}) \times (M_p \times \cdots \times M_{n-1})$$

i.e., the final matrix is obtained by multiplying a matrix of dimension $(w_1 \times w_p)$ and a matrix of dimension $(w_p \times w_n)$. Then in the partition of the n-gon, we let the triangle with vertices V_1 and V_n have the third vertex V_p. The polygon $V_1 - V_2 - \cdots - V_p$ is a convex polygon of p sides with base $V_1 - V_p$ and its partition corresponds to an order of multiplying matrices $M_1, \ldots, M_{p-1}$, giving a matrix of dimension $w_1 \times w_p$. Similarly, the partition of the polygon $V_p - V_{p+1} - \cdots - V_n$ with base $V_p - V_n$ corresponds to an order of multiplying matrices $M_p, \ldots, M_{n-1}$, giving a matrix of dimension $w_p \times w_n$. Hence the triangle $V_1 V_p V_n$ with base $V_1 - V_n$ represents the multiplication of the two partial products, giving the final matrix of dimension $w_1 \times w_n$. ■

<u>Lemma 2.</u> The minimum number of operations to evaluate the following matrix chain products are identical.

$$M_1 \times M_2 \times \cdots \times M_{n-2} \times M_{n-1}$$

$$M_n \times M_1 \times \cdots \times M_{n-3} \times M_{n-2}$$

$$\vdots$$

$$M_2 \times M_3 \times \cdots \times M_{n-1} \times M_n$$

where M_i has dimension $w_i \times w_{i+1}$ and $w_{n+1} \equiv w_1$. Note that in the first matrix chain, the resulting matrix is of dimension w_1 by w_n. In the last matrix chain, the resulting matrix is of dimension w_2 by w_1. But in all the cases, the total number of operations in the optimum orders of multiplication is the same.

Proof. The cyclic permutations of the n-1 matrices all correspond to the same n-gon and thus have the same optimum partitions. ■

(This Lemma was obtained independently in [4] with a long proof.)

From now on, we shall concentrate only on the partitioning problem.

The diagonals inside the polygon are called arcs. Thus, one easily verifies inductively that every partition consists of n-2 triangles formed by n-3 arcs and n sides.

In a partition of an n-gon, the degree of a vertex is the number of arcs incident on the vertex plus two (since there are two sides incident on every vertex).

Lemma 3. In any partition of an n-gon, $n \geq 4$, there are at least two triangles, each having a vertex of degree two. (For example, in Fig. 1, the triangle $V_1V_2V_3$ has vertex V_2 with degree 2 and the triangle $V_4V_5V_6$ has vertex V_5 with degree 2.) (See also [5].)

Proof. In any partition of an n-gon, there are n-2 non-intersecting triangles formed by n-3 arcs and n sides. And for any $n \geq 4$, no triangle can be formed by 3 sides. Let x be the number of triangles with two sides and one arc, y be the number of triangles with one side and two arcs, and z be the number of triangles with three arcs. Since an arc is used in two triangles, we have

$$x + 2y + 3z = 2(n-3) \quad . \tag{4}$$

Since the polygon has n sides, we have

$$2x + y = n \quad . \tag{5}$$

From (4) and (5), we get

$$3x = 3z + 6 \quad .$$

Since $z \geq 0$, we have $x \geq 2$. ■

Lemma 4. Let P and P' both be n-gons where the corresponding weights of the vertices satisfy $w_i \leq w_i'$, then the cost of an optimum partition of P is less than or equal to the cost of an optimum partition of P'.

Proof. Omitted. ■

If we use $C(w_1, w_2, w_3, \ldots, w_k)$ to mean the minimum cost of partitioning the k-gon with weights w_i optimally, Lemma 4 can be stated as

$$C(w_1, w_2, \ldots, w_k) \leq C(w_1', w_2', \ldots, w_k') \text{ if } w_i \leq w_i' \; .$$

We say that two vertices are connected in an optimum partition if the two vertices are connected by an arc or if the two vertices are adjacent to the same side.

In the rest of the paper, we shall use $V_1, V_2, \ldots, V_n$ to denote vertices which are ordered according to their weights, i.e. $w_1 \leq w_2 \leq \cdots \leq w_n$. To facilitate the presentation, we introduce a tie-breaking rule for vertices of equal weights.

If there are two or more vertices with weights equal to the smallest weight w_1, we can arbitrarily choose one of these vertices to be the vertex V_1. Once the vertex V_1 is chosen, further ties in equal weights are resolved by regarding the vertex which is closer to V_1 in the clockwise direction to be of less weight. With this tie-breaking rule, we can unambiguously label the vertices $V_1, V_2, \ldots, V_n$ for each choice of V_1. A vertex V_i is said to be smaller than another vertex V_j, denoted by $V_i < V_j$, either if $w_i < w_j$ or if $w_i = w_j$ and $i < j$. We say that V_i is the smallest vertex in a subpolygon if it is smaller than any other vertices in the subpolygon.

After the vertices are labeled, we define an arc V_i-V_j to be less than another arc V_p-V_q

$$\text{if} \quad \min(i,j) < \min(p,q)$$

$$\text{or} \quad \begin{cases} \min(i,j) = \min(p,q) \\ \max(i,j) < \max(p,q) \end{cases} .$$

(For example, the arc V_3-V_9 is less than the arc V_4-V_5.) Every partition of an n-gon has n-3 arcs which can be sorted from the smallest to the largest into an ordered sequence of arcs, i.e., each partition is associated with a unique ordered sequence of arcs. We define a partition P to be lexicographically less than a partition Q if the ordered sequence of arcs associated with P is lexicographically less than that associated with Q.

When there is more than one optimum partition, we use the <u>l-optimum partition</u> (i.e., lexicographically-optimum partition) to mean the lexicographically smallest optimum partition, and use <u>an optimum partition</u> to mean some partition of minimum cost.

We shall use $V_a, V_b, \ldots$ to denote vertices which are unordered in weights, and T_{ijk} to denote the product of the weights of any three vertices V_i, V_j and V_k.

<u>Theorem 1.</u> For every way of choosing $V_1, V_2, \ldots$ (as prescribed), there is always an optimum partition containing V_1-V_2 and V_1-V_3. (Here, V_1-V_2 and V_1-V_3 may be either arcs or sides.)

<u>Proof:</u> The proof is by induction. For the optimum partitions of a triangle and a 4-gon, the theorem is true. Assume that the theorem is true for all k-gons ($3 \le k \le n-1$) and consider the optimum partitions of an n-gon.

From Lemma 3, in any optimum partition, we can find at least two vertices having degree two. Call these two vertices V_i and V_j. We can divide this into two cases.

(i) One of the two vertices V_i (or V_j) is not V_1, V_2 or V_3 in some optimum partition of the n-gon. In this case, we can remove the vertex V_i with its two sides and obtain an (n-1)-gon. In this (n-1)-gon, V_1, V_2, V_3 are the three vertices with smallest weights. By the induction assumption, V_1 is connected to both V_2 and V_3 in an optimum partition.

(ii) Consider the complementary case of (i), in all the optimum partitions of the n-gon, all the vertices with degree two are from the set $\{V_1, V_2, V_3\}$. (In this case, there will be at most three vertices with degree two in every optimum partition.) We have the following three subcases:

(a) $V_i = V_2$ and $V_j = V_3$ in some optimum partition of the n-gon, i.e., both V_2 and V_3 have degree two simultaneously. In this case, we first remove V_2 with its two sides and form an (n-1)-gon. By the induction assumption, V_1, V_3 must be connected in some optimum partition. If V_1-V_3 appears as an arc, it reduces to (i). So V_1-V_3 must appear as a side of the (n-1)-gon, and reattaching V_2 to the (n-1)-gon shows that either V_1, V_2 and V_3 are mutually adjacent or V_1-V_3 is a side of the n-gon. In the former case, the proof is complete, so we assume that V_1-V_3 is a side of the n-gon. Similarly, we can remove V_3 with its two sides and show that V_1, V_2 are connected by a side of the n-gon.

(b) $V_i = V_1$ and $V_j = V_2$ in some optimum partition of the n-gon, i.e., V_1 and V_2 both have degree two simultaneously. In this case, we can first remove V_1 and form an (n-1)-gon where V_2, V_3, V_4 are the three vertices with smallest weights. By the induction assumption, V_2 is connected to both V_3 and V_4 in an optimum partition. If V_2-V_3 or V_2-V_4 appears as an arc, it reduces to (i). Hence, V_2-V_3 and V_2-V_4 must both be sides of the n-gon. Similarly, we can remove V_2 with its two sides and form an (n-1)-gon where V_1, V_3, V_4 are the three vertices with smallest weights.

Again, V_1 must be connected to V_3 and V_4 by sides of the n-gon. But for any n-gon with $n \geq 5$, it is impossible to have V_3 and V_4 both adjacent to V_1 and V_2 at the same time, i.e., V_1 and V_2 cannot both have degree two in an optimum partition of any n-gon with $n \geq 5$.

(c) $V_i = V_1$, $V_j = V_3$ in some optimum partition of the n-gon. By argument similar to (b), we can show that V_2 must be adjacent to V_1 and V_3 in the n-gon. The situation is as shown in Fig. 3(a). Then the partition in Fig. 3(b) is cheaper because

$$T_{123} \leq T_{12q}$$

and $C(w_1, w_q, w_y, w_t, w_x, w_p, w_3) \leq C(w_2, w_q, w_y, w_t, w_x, w_p, w_3)$

according to Lemma 4. ■

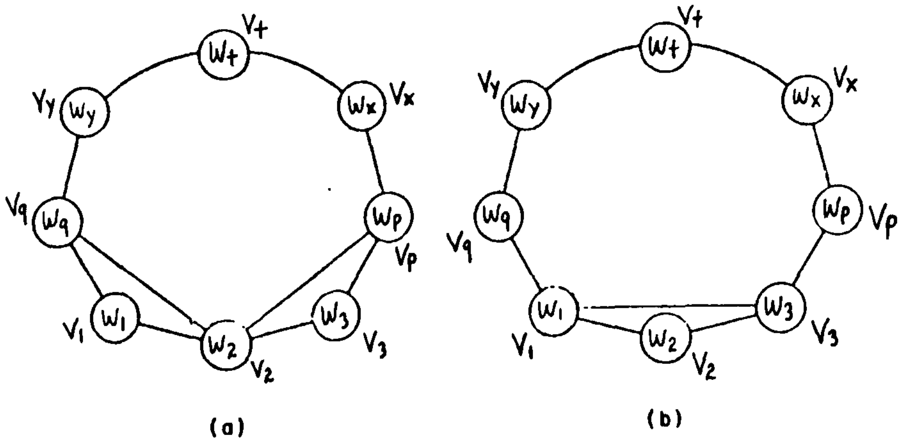


Fig. 3

Corollary 1. For every way of choosing $V_1, V_2, \ldots$ (as prescribed), the ℓ-optimum partition always contains V_1-V_2 and V_1-V_3.

Proof: It follows from Theorem 1 and the definition of the ℓ-optimum partition. ■

Once we know V_1-V_2 and V_1-V_3 always exist in the ℓ-optimum partition, we can use this fact recursively. Hence, in finding the ℓ-optimum partition of a given polygon, we can decompose it into subpolygons by joining the smallest vertex with the second smallest and third smallest vertices repeatedly, until each of these subpolygons has the property that its smallest vertex is adjacent to both its second smallest and the third smallest vertices.

A polygon having V_1 adjacent to V_2 and V_3 by sides will be called a basic polygon.

Theorem 2. A necessary but not sufficient condition for V_2-V_3 to exist in an optimum partition of a basic polygon is

$$\frac{1}{w_1} + \frac{1}{w_4} \le \frac{1}{w_2} + \frac{1}{w_3}$$

Furthermore, if V_2-V_3 is not present in the ℓ-optimum partition, then V_1, V_4 are always connected in the ℓ-optimum partition.

Proof. If V_2, V_3 are not connected in the ℓ-optimum partition of a basic polygon, the degree of V_1 is greater than or equal to 3. Let V_p be a vertex in the polygon and V_1, V_p are connected in the ℓ-optimum partition. V_4 is either in the subpolygon containing V_1, V_2 and V_p or in the subpolygon containing V_1, V_3 and V_p. In either case, V_4 will be the third smallest vertex in the subpolygon. From Corollary 1, V_1, V_4 are connected in the ℓ-optimum partition of the subpolygon and it also follows that V_1, V_4 are connected in the ℓ-optimum partition of the basic polygon.

If V_2, V_3 are connected in an optimum partition, then we have an (n-1)-gon where V_2 is the smallest vertex and V_4 is the third smallest vertex. By Theorem 1, there exists an optimum partition of the (n-1)-gon in which V_2, V_4 are connected. Thus by induction on n, we can assume that V_4 is adjacent to V_2 in the basic polygon as shown in Fig. 4(a).

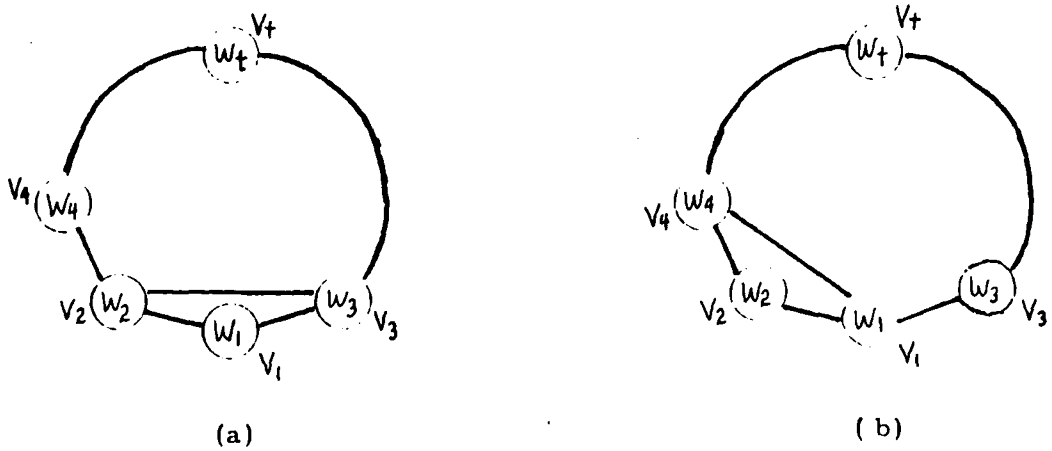


Fig. 4

The cost of the partition in Fig. 4(a) is

$$T_{123} + C(w_2, w_4, \ldots, w_t, \ldots, w_3) \quad . \tag{7}$$

And the cost of the partition in Fig. 4(b) is

$$T_{124} + C(w_1, w_4, \ldots, w_t, \ldots, w_3) \quad . \tag{8}$$

According to Lemma 4,

$$C(w_1, w_4, \ldots, w_t, \ldots, w_3) < C(w_2, w_4, \ldots, w_t, \ldots, w_3) \quad . \tag{9}$$

Since the weights of the vertices between V_4 and V_3 in the clockwise direction are all greater than or equal to w_4, the difference between RHS and LHS of (9) is at least

$$T_{243} - T_{143} \quad .$$

So the necessary condition for (7) to be no greater than (8) is

$$T_{123} + T_{243} \le T_{124} + T_{134}$$

or

$$\frac{1}{w_1} + \frac{1}{w_4} \le \frac{1}{w_2} + \frac{1}{w_3} \quad . \qquad \blacksquare$$

<u>Lemma 5</u>. In an optimum partition of an n-gon, let V_x, V_y, V_z, and V_w be four vertices of an inscribed quadrilateral (V_x and V_z are not adjacent in the quadrilateral). A necessary condition for V_x-V_z to exist is

$$\frac{1}{w_x} + \frac{1}{w_z} \ge \frac{1}{w_y} + \frac{1}{w_w} \quad . \tag{10}$$

<u>Proof</u>: The cost of partitioning the quadrilateral by the arc V_x-V_z is

$$T_{xyz} + T_{xzw} \tag{11}$$

and the cost of partitioning the quadrilateral by the arc V_y-V_w is

$$T_{xyw} + T_{yzw} \quad . \tag{12}$$

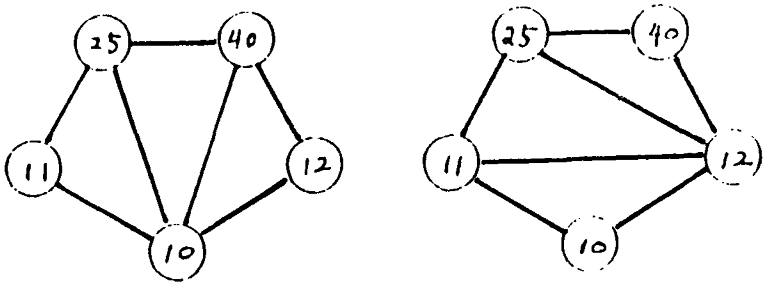
For optimality, we have (11) $\le$ (12) which is (10). $\blacksquare$

Note that if strict inequality holds in (10), the necessary condition is also sufficient. If equality holds in (10), the condition is sufficient for V_x-V_z to exist in the ℓ-optimum partition provided $\min(x, z) < \min(y, w)$. This lemma is a generalization of Lemma 1 of Chin [3] where V_y is the vertex with the smallest weight and V_x, V_w, V_z are three consecutive vertices with w_w greater than both w_x and w_z.

A partition is called stable if every quadrilateral in the partition satisfies (10).

Corollary 2. An optimum partition is stable but a stable partition may not be optimum.

Proof. The fact that optimum partition has to be stable follows from Lemma 5. Figure 5 gives an example that a stable partition may not be optimum. ■



(a) a stable partition (b) the optimum partition

Fig. 5

In any partition of an n-gon, every arc dissects a unique quadrilateral. Let V_x, V_y, V_z, V_w be the four vertices of an inscribed quadrilateral and V_x-V_z be the arc which dissects the quadrilateral. We define V_x-V_z to be a vertical arc if (13) or (14) is satisfied.

$$\min(w_x, w_z) < \min(w_y, w_w) \tag{13}$$

$$\left.\begin{aligned} \min(w_x, w_z) &= \min(w_y, w_w) \\ \max(w_x, w_z) &\le \max(w_y, w_w) \end{aligned}\right\} \tag{14}$$

We define V_x-V_z to be a horizontal arc if (15) is satisfied

$$\left.\begin{aligned} \min(w_x, w_z) &> \min(w_y, w_w) \\ \max(w_x, w_z) &< \max(w_y, w_w) \end{aligned}\right\} \tag{15}$$

For brevity, we shall use h-arcs and v-arcs to denote horizontal arcs and vertical arcs from now on.

Corollary 3. All arcs in an optimum partition must be either vertical arcs or horizontal arcs.

Proof: Let V_x-V_z be an arc which is neither vertical nor horizontal. There are two cases:

Case 1. $\min(w_x, w_z) = \min(w_y, w_w)$

and $\max(w_x, w_z) > \max(w_y, w_w)$

Case 2. $\min(w_x, w_z) > \min(w_y, w_w)$

and $\max(w_x, w_z) \ge \max(w_y, w_w)$.

In both cases, the inequality (10) in Lemma 5 cannot be satisfied. This implies that the partition is not stable and hence cannot be optimum. ■

Theorem 3. Let V_x and V_z be two arbitrary vertices which are not adjacent in a polygon, and V_w be the smallest vertex from V_x to V_z in the clockwise manner ($V_w \neq V_x$, $V_w \neq V_z$), and V_y be the smallest vertex from V_z to V_x in the clockwise manner ($V_y \neq V_x$, $V_y \neq V_z$). This is shown in Fig. 6 where without loss of generality, we assume that $V_x < V_z$ and $V_y < V_w$. A necessary condition for $V_x - V_z$ to exist as an h-arc in the ℓ-optimum partition is that

$$w_y < w_x \leq w_z < w_w \quad .$$

(Note that the necessary condition still holds when the positions of V_y and V_w are interchanged.)

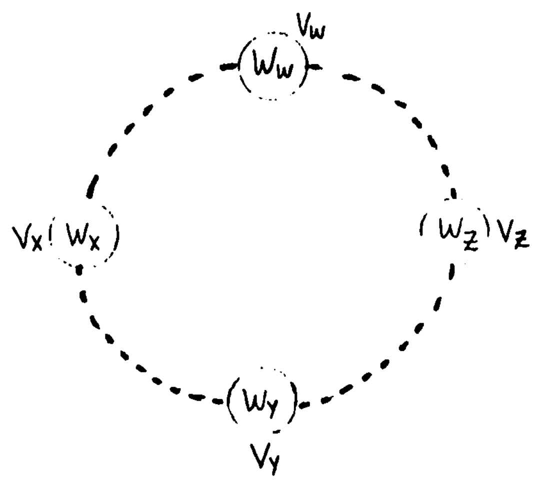


Fig. 6

Proof. The proof is by contradiction. If $w_x \leq w_y$, w_x must be equal to the smallest weight w_1 and V_x-V_z can never satisfy (15). Hence, in order that V_x-V_z exists as an h-arc in the ℓ-optimum partition, we must have $w_y < w_x \leq w_z$. Since V_y is the smallest vertex from V_z to V_x in the clockwise manner and $V_x < V_w$, we must have $V_y = V_1$.

Assume for the moment that $V_3 < V_x < V_z$. From Corollary 1, both V_1-V_2 and V_1-V_3 exist in the ℓ-optimum partition, and the two arcs would divide the polygon into subpolygons. If V_x and V_z are in different subpolygons, then they cannot be connected in the ℓ-optimum partition. Without loss of generality, we can assume that the polygon is a basic polygon. In this basic polygon, either V_2-V_3 or V_1-V_4 exists in the ℓ-optimum partition (Theorem 2).

If V_2, V_3 are connected, then V_x and V_z are both in a smaller polygon in which we can treat V_2 as the smallest vertex and repeat the argument. If V_1, V_4 are connected, the basic polygon is again divided into two subpolygons and V_x and V_z both have to be in one of the subpolygons and the subpolygon has at most n-1 sides. (Otherwise V_x-V_z can never exist in the ℓ-optimum partition.) The successive reduction in the size of the polygon will either make the connection V_x-V_z impossible, or force V_x and V_z to become the second smallest and the third smallest vertices in a basic subpolygon. Let V_m be the smallest vertex in this basic subpolygon. In order that V_x-V_z appear as an h-arc, we must have $w_x > w_m$. From Theorem 2, the necessary condition for V_x-V_z (i.e. V_2-V_3) to exist in an optimum partition of the subpolygon is

$$\frac{1}{w_x} + \frac{1}{w_z} \geq \frac{1}{w_m} + \frac{1}{w_w} .$$

Since $w_x > w_m$, the inequality is valid only if $w_z < w_w$. ∎

Corollary 4. A weaker necessary condition for V_x-V_z to exist as an h-arc in the ℓ-optimum partition is that

$$V_y < V_x < V_z < V_w .$$

Proof. This follows from Theorem 3. ∎

We call any arc which satisfies this weaker necessary condition a potential h-arc. Let P be the set of potential h-arcs in the n-gon and H be the set of h-arcs in the ℓ-optimum partition, we have $P \supseteq H$ where the inclusion could be proper.

Corollary 5. Let V_w be the largest vertex in the polygon and V_x and V_z be its two neighboring vertices. If there exists a vertex V_y such that $V_y < V_x$ and $V_y < V_z$, then V_x-V_z is a potential h-arc.

Proof. This follows directly from Corollary 4 where there is only one vertex between V_x and V_z. ■

Two arcs are called compatible if both arcs can exist simultaneously in a partition. Assume that all weights of the vertices are distinct, then there are (n-1)! distinct permutations of the weights around an n-gon. For example, the weights 10, 11, 25, 40, 12 in Fig. 5(a) correspond to the permutation w_1, w_2, w_4, w_5, w_3 (where $w_1 < w_2 < w_3 < w_4 < w_5$). There are infinitely many values of the weights which correspond to the same permutation. For example, 1, 16, 34, 77, 29 also corresponds to w_1, w_2, w_4, w_5, w_3 but its optimum partition is different from that of 10, 11, 25, 40, 12. However, all the potential h-arcs in all the n-gons with the same permutation of weights are compatible. We state this remarkable fact as Theorem 4.

Theorem 4. All potential h-arcs are compatible.

Proof. The proof is by contradiction. Let V_x, V_y, V_z and V_w be the four vertices described in Theorem 3. Hence, we have $V_y < V_x < V_z < V_w$

and V_x-V_z is a potential h-arc. Let V_p-V_q be a potential h-arc which is not compatible to V_x-V_z, as shown in Fig. 7. Without loss of generality, we can assume $V_p < V_q$. (The proof for the case $V_q < V_p$ is similar to that which follows.)

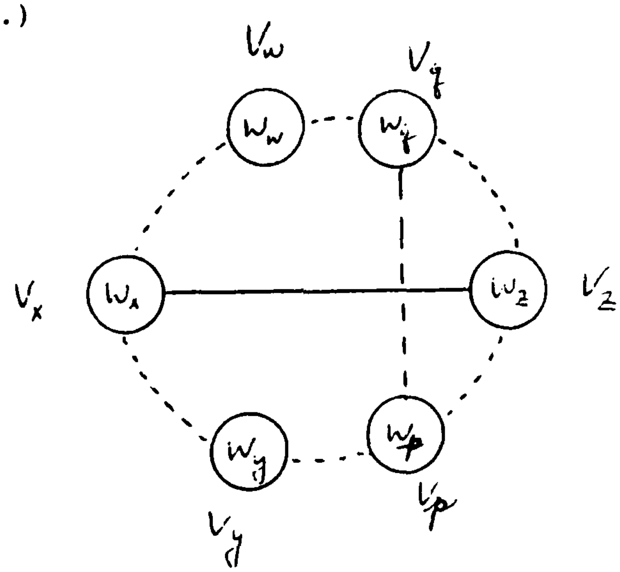


Fig. 7

Since V_w is the smallest vertex between V_x and V_z in the clockwise manner, we have $V_z < V_w < V_q$. Hence, we have either $V_y < V_p < V_z < V_q$ or $V_y < V_z < V_p < V_q$. Both cases violate Corollary 4 and V_p - V_q cannot be a potential h-arc. ■

Note that the potential h-arc V_x-V_z always dissects the n-gon into two subpolygons and one of these subpolygons has the property that all its vertices except V_x and V_z have weights no smaller than $\max(w_x, w_z)$. We shall call this subpolygon the <u>upper</u> subpolygon of V_x-V_z. For example, the subpolygon V_x - . . . . - V_w - . . . . - V_q - . . . . - V_z in Fig. 7 is the upper subpolygon of V_x-V_z.

Using Corollary 4 and Theorem 4, we can generate all the potential h-arcs of a polygon.

Let V_x-V_z be the arc defined in Corollary 5, i.e. $V_1 < V_x < V_z < V_w$. The arc V_x-V_z is a potential h-arc compatible to all other potential h-arcs in the n-gon. Furthermore, there is no other potential h-arc in its upper subpolygon. Now consider the (n-1)-gon obtained by cutting out V_w. In this (n-1)-gon, let $V_{w'}$ be the largest vertex and $V_{x'}$ and $V_{z'}$ be the two neighbors of $V_{w'}$ where $V_1 < V_{x'} < V_{z'} < V_{w'}$. Then $V_{x'}$-$V_{z'}$ is again a potential h-arc compatible to all other potential h-arcs in the n-gon and there is no other potential h-arc in its upper subpolygon which has not been generated. This is true even if V_w is in the upper subpolygon of $V_{x'}$-$V_{z'}$. If we repeat the process of cutting out the largest vertex, we get a set P of arcs, all arcs satisfy Corollary 4. The h-arcs of the ℓ-optimum partition must be a subset of these arcs.

The process of cutting out the largest vertex can be made into an algorithm which is O(n). We shall call this algorithm the <u>one-sweep algorithm.</u> The output of the one-sweep algorithm is a set S of n-3 arcs. S is empty initially.

<u>The one-sweep algorithm:</u>

Starting from the smallest vertex, say V_1, we travel in the clockwise direction around the polygon and push the weights of the vertices successively onto the stack as follows (w_1 will be at the bottom of the stack).

(a) Let V_t be the top element on the stack, V_{t-1} be the element immediately below V_t, and V_c be the element to be pushed onto the stack.

If there are two or more vertices on the stack and $w_t > w_c$, add V_{t-1}-V_c to S, pop V_t off the stack; if there is only one vertex on the stack or $w_t \leq w_c$, push w_c onto the stack. Repeat this step until the n^{th} vertex has been pushed onto the stack.

(b) If there are more than three vertices on the stack, add V_{t-1}-V_c to S, pop V_t off the stack and repeat this step, else stop.

Since we do not check for the existence of a smallest vertex whose weight is strictly no larger than those of the two neighbors of the largest vertex, i.e. the existence of the vertex V_y in Corollary 4, not all the n-3 arcs generated by the algorithm are potential h-arcs. However, it is not difficult to verify that the one-sweep algorithm always generates a set S of n-3 arcs which contains *the set* P *of all potential* h-arcs which contains the set H of all h-arcs in the ℓ-optimum partition of the n-gon, i.e.,

$$S \supseteq P \supseteq H$$

where each inclusion could be proper. For example, if the weights of the vertices around the n-gon in the clockwise direction are $w_1, w_2, \ldots, w_n$ where $w_1 \leq w_2 \leq \cdots \leq w_n$, none of the arcs in the n-gon can satisfy Corollary 4 and hence there are no potential h-arcs in the n-gon. The one-sweep algorithm would still generate n-3 arcs for the n-gon but none of the arcs generated is a potential h-arc.

3. Conclusion

In this paper, we have presented several theorems on the Polygon Partitioning Problem. Some of these theorems are characterizations of the optimum partitions of any n-sided convex polygon, while the others apply to the unique lexicographically smallest optimum partition. Based on these theorems, an O(n) algorithm for finding a near-optimum partition can be developed [12]. The cost of the partition produced by the heuristic algorithm never exceeds 1.155 Copt, where Copt is the optimum cost of partitioning the polygon. An O(n log n) algorithm for finding the unique lexicographically smallest optimum partition will be presented in part II.

4. Acknowledgment

The authors would like to thank the referees for their helpful comments in revising the manuscript.

References

1. A. V. Aho, J. E. Hopcroft and J. D. Ullman, "The Design and Analysis of Computer Algorithms," Addison-Wesley, 1974.

2. A. K. Chandra, "Computing Matrix Chain Product in Near Optimum Time," IBM Res. Report RC5626 (#24393), IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1975.

3. F. Y. Chin, "An O(n) Algorithm for Determining a Near Optimal Computation Order of Matrix Chain Product," Communication of ACM, Vol. 21, No. 7, July 1978, pp. 544-549.

4. L. E. Deimel, Jr. and T. A. Lampe, "An Invariance Theorem Concerning Optimal Computation of Matrix Chain Products," North Carolina State Univ. Report TR79-14.

5. G. A. Dirac, "On rigid circuit graphs," Abh. Math. Sem., Univ. Hamburg, 25, pp. 71-76 (1961).

6. M. Gardner, "Catalan numbers," Scientific American, June 1976, pp. 120-124.

7. S. S. Godbole, "An Efficient Computation of Matrix Chain Products," IEEE Trans. Computers C-22, 9 Sept. 1973, pp. 864-866.

8. H. W. Gould, "Bell and Catalan Numbers," Combinatorial Research Institute, Morgantown, W. Va., June 1977.

9. T. C. Hu and M. T. Shing, "Computation of Matrix Chain Product," Abstract, Am. Math. Soc., Vol. 1, No. 3, p. 336, April 1980.

10. T. C. Hu and M. T. Shing, "Some Theorems about Matrix Multiplications," Proceedings of the 21st Annual IEEE Symposium on the Foundation of Computer Science, Oct. 1980, pp. 28-35.

11. T. C. Hu and M. T. Shing, "Computation of Matrix Chain Products," Proceedings of 1981 Army Numerical Analysis & Computations Conference, August 1981, pp. 615-628.

12. T. C. Hu and M. T. Shing, "An O(n) Algorithm to Find a Near-Optimum Partition of a Convex Polygon," to appear in the Journal of Algorithms.

Rev. August 10, 1981

Computation of Matrix Chain Products, Part II

T. C. Hu and M. T. Shing

University of California, San Diego

La Jolla, CA 92093

Abstract:

This paper considers the computation of matrix chain products of the form $M_1 \times M_2 \times \ldots \times M_{n-1}$. If the matrices are of different dimensions, the order in which the matrices are computed affects the number of operations. An optimum order is an order which minimizes the total number of operations. Some theorems about an optimum order of computing the matrices have been presented in part I. Based on those theorems, an O(n log n) algorithm for finding the optimum order is presented here.

This research was supported in part by National Science Foundation grant MCS-77-23738 and U.S. Army Research Office grant DAAG29-80-C-0029.

1. Introduction

In Part I of this paper, we have transformed the matrix chain product problem into the optimum partitioning problem and have stated several theorems about the optimum partitions of an n-sided convex polygon. Based on these theorems, we now present algorithms for finding the unique ℓ-optimum (lexicographically smallest optimum) partition.

Using the same notation as in Part I of this paper, we can assume that we have uniquely labelled all vertices of the n-gon. A partition is called a fan if it consists of only v-arcs joining the smallest vertex to all other vertices in the polygon. We shall denote the fan of a polygon $V_1 - V_b - V_c - \cdots - V_n$ by $\mathrm{Fan}(w_1 | w_b, w_c, \ldots, w_n)$. The smallest vertex V_1 is called the center of the fan.

We define a vertex as a local maximum vertex if it is larger than its two neighbors and define a vertex as a local minimum vertex if it is smaller than its two neighbors. A polygon is called a monotone polygon if there exists only one local maximum and one local minimum vertex. We shall first give an O(n) algorithm for finding the ℓ-optimum partition of a monotone polygon and then give an O(n log n) algorithm for finding the ℓ-optimum partition of a general convex polygon.

2. Monotone Basic Polygon

In this section, let us consider the optimum partition of a monotone polygon, i.e. a polygon with only one local minimum vertex and one local maximum vertex. It follows from Corollary 1 of Part I that we can

consider a monotone basic polygon only. The understanding of this special case is necessary in finding the optimum partition of a general convex polygon.

Consider a monotone basic n-gon V_1-V_2-V_c-$\cdots$-V_3, the fan of the polygon is denoted by

$$\mathrm{Fan}(w_1 | w_2, w_c, \ldots, w_3)$$

where the smallest vertex V_1 is the center of the fan.

The definition of a fan can also be applied to subpolygons as well. For example, if V_2, V_3 are connected in the basic n-gon and V_2 becomes the smallest vertex in the (n-1)-sided subpolygon, the partition formed by connecting V_2 to all vertices in the (n-1)-gon is denoted by

$$\mathrm{Fan}(w_2 | w_c, \ldots, w_3) \ .$$

Lemma 1. If none of the potential h-arcs appears in the ℓ-optimum partition of the n-gon, the ℓ-optimum partition must be the fan of the n-gon.

Proof. From Theorem 3 of Part I, we know that any arc which exists as an h-arc in the ℓ-optimum partition mus ɔe a potential h-arc. Hence, if the ℓ-optimum partition does not contain any potential h-arc, the ℓ-optimum partition must be made up of v-arcs only. Hence, we ıave to show that among all partitions which are made up of v-arcs only, the fan is (i) the lexicographically smallest and (ii) one of the cheapest partitions in the n-gon.

(i) Since the fan consists of only v-arcs joining V_1 to all other vertices in the n-gon, it is by definition the lexicographically smallest partition.

(ii) Suppose the ℓ-optimum partition contains v-arcs only but is not the fan. There must exist three vertices V_i, V_k, V_j such that the triangles

$V_1V_iV_j$ and $V_iV_jV_k$ are present in the ℓ-optimum partition. Since V_i-V_j is a v-arc (by assumption) and V_1 is the smallest vertex in the n-gon, we have $w_1 = \min(w_i, w_j)$ and $\max(w_i, w_j) \leq w_k$. If we replace the v-arc V_i-V_j by the v-arc V_1-V_k, we can get a partition whose cost is less than or equal to that of the ℓ-optimum partition but is lexcographically smaller than the ℓ-optimum partition, and results in a contradiction. ■

Let V_i-V_j and V_p-V_q be two potential h-arcs of any n-gon. We say that V_i-V_j is <u>above</u> V_p-V_q (and V_p-V_q is <u>below</u> V_i-V_j) if the upper subpolygon of V_p-V_q contains the upper subpolygon of V_i-V_j.

Let P be the set of all potential h-arcs in a monotone basic n-gon. P can have at most (n-3) arcs.

<u>Lemma 2</u>. For any two arcs in P, say V_i-V_j and V_p-V_q, we must have either V_i-V_j above V_p-V_q or V_p-V_q above V_i-V_j.

<u>Proof</u>. By contradiction. Let V_i-V_j and V_p-V_q be two arcs in P which do not satisfy this lemma. Then the intersection of the upper subpolygons of V_i-V_j and V_p-V_q must either be empty or consists of part of each upper subpolygon only.

Since the vertices other than V_i, V_j in the upper subpolygon of V_i-V_j must have weights larger than $\max(w_i, w_j)$, the local maximum vertex of the monotone basic polygon must be present in the upper subpolygon of V_i-V_j. Similarly, the local maximum vertex of the monotone basic polygon must also be present in the upper subpolygon of V_p-V_q. Hence, the intersections of the upper subpolygons of V_i-V_j and V_p-V_q cannot be empty.

From Theorem 4 of Part I, we know that V_i-V_j and V_p-V_q cannot cross each other and hence the intersection of their upper subpolygons cannot consist of part of each upper subpolygons only. ■

We can actually show this ordering of potential h-arcs pictorially by drawing a monotone basic polygon in such a way that the local maximum vertex is always at the top and the local minimum vertex is at the bottom. Then a potential h-arc V_i-V_j is physically above another potential h-arc V_p-V_q if the upper subpolygon of V_p-V_q contains the upper subpolygon of V_i-V_j. From the definition of the upper subpolygon, we can see that $\min(w_i, w_j) > \max(w_p, w_q)$ if V_i-V_j is above V_p-V_q.

Consider the monotone basic n-gon which is shown symbolically in Figure 1. V_n is the local maximum vertex and V_i-V_j, V_p-V_q are potential h-arcs of the monotone basic n-gon. The subpolygon V_p-$\cdots$-V_i-V_j-$\cdots$-V_q which is formed by two potential h-arcs V_p-V_q and V_i-V_j and the sides of the n-gon from V_p to V_i and from V_j to V_q in the clockwise direction is said to be bounded above by the potential h-arc V_i-V_j and bounded below by the potential h-arc V_p-V_q.

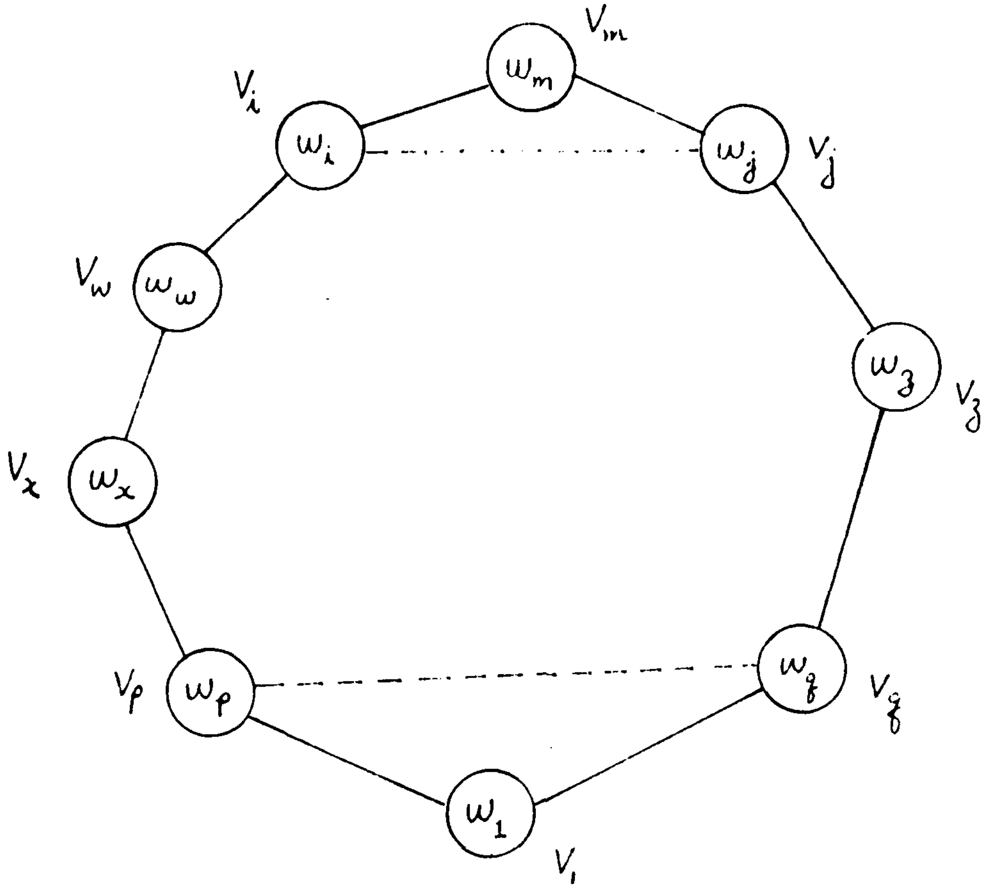


Figure 1

Lemma 3. Any subpolygon which is bounded by two potential h-arcs of the monotone basic n-gon is itself a monotone polygon.

Proof. Consider the subpolygon $V_p - \cdots - V_i - V_j - \cdots - V_q$ in Figure 1. Without loss of generality, we can assume $V_i < V_j$ and $V_p < V_q$. Since V_n is the only local maximum vertex in the monotone basic n-gon, we must have $V_1 < V_p < \cdots < V_i < V_n$ and $V_n > V_j > \cdots > V_q > V_1$. Hence, V_p is the unique local minimum vertex and V_j is the unique local maximum vertex in the subpolygon $V_p - \cdots - V_i - V_j - \cdots - V_q$. By definition, $V_p - \cdots - V_i - V_j - \cdots - V_q$ is a monotone polygon. ■

Lemma 4. Any potential h-arc of a subpolygon bounded above and below by two potential h-arcs of the monotone basic n-gon is also a potential h-arc of the monotone basic n-gon.

Proof. Consider the subpolygon V_p-$\cdots$-V_i-V_j-$\cdots$-V_q in Figure 1. Let V_x-V_z be a potential h-arc in this subpolygon and V_w is the smallest vertex between V_x and V_z in the clockwise direction around the subpolygon. Without loss of generality, we can assume $V_i < V_j$, $V_p < V_q$ and $V_x < V_z$. Since V_x is in the upper subpolygon of the potential h-arc V_p-V_q, we have $w_1 < w_p \leq w_q < w_x \leq w_z$. Since $V_j <$ any vertex in the upper subpolygon of V_i-V_j and $V_w < V_i < V_j$, V_w is the smallest vertex between V_x and V_z in clockwise direction around the monotone basic n-gon. Hence, we have $w_1 < w_x \leq w_z < w_w$ and V_x-V_z is a potential h-arc of the monotone basic n-gon. ■

We can now summarize what we have discussed. If there is no h-arc in the ℓ-optimum partition of a monotone basic n-gon, the ℓ-optimum partition must be a fan. Otherwise, the h-arcs in the ℓ-optimum partition are all layered, one above another. If we consider the local maximum vertex V_n and the local minimum vertex V_1 as two degenerated h-arcs, then the ℓ-optimum partition of a monotone basic n-gon will contain one or more monotone subpolygons, each bounded above and below by two h-arcs and the ℓ-optimum partition of each of these monotone subpolygons is a fan.

Then, in finding the ℓ-optimum partition of a monotone basic polygon, we have only to consider those partitions which contain one or more subpolygons bounded above and below by potential h-arcs and each of these subpolygons is partitioned by a fan. Since there are at most (n-3) non-degenerated potential h-arcs in a monotone basic n-gon, there will be at most 2^{n-3} such partitions and we can divide all these partitions into (n-2) classes by the number of non-degenerated potential h-arcs a partition contains. These classes are denoted by $H_0, H_1, \ldots, H_{n-3}$ where the subscript indicates the number of non-degenerated potential h-arcs in each partition of that class.

There is no potential h-arc in the partitions in the class H_o. Hence the class consists of only one partition, namely the fan

$$\mathrm{Fan}(w_1 | w_2, \ldots, w_3).$$

In the class H_1, each partition has one non-degenerated potential h-arc. Once the potential h-arc is known, the rest of the arcs must all be vertical arcs forming two fans, one in each subpolygon.

Two typical partitions in H_1 of a monotone basic polygon are shown in Fig. 2. In Fig. 2a, there is one non-degenerated potential h-arc, $V_c - V_i$ ($V_c < V_i$). The upper subpolygon is a fan

$$\mathrm{Fan}(w_c | w_d, \ldots, w_i)$$

and the lower subpolygon is a fan

$$\mathrm{Fan}(w_1 | w_2, w_c, w_i, w_3) \ .$$

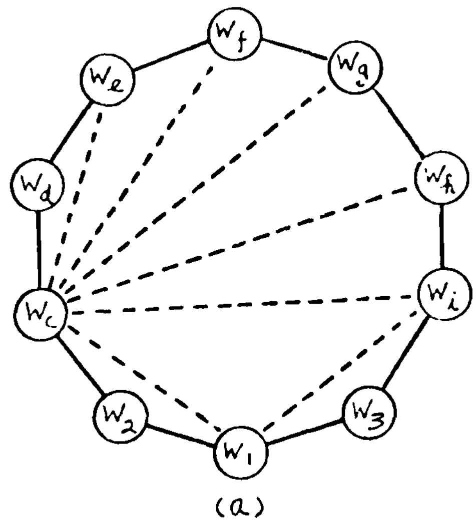


Fig. 2. Two typical partitions in H_1 of a monoton 10-gon.

In Fig. 2b, there is one potential h-arc, V_2-V_3, and the upper subpolygon is a fan

$$\mathrm{Fan}(w_2 | w_c, \ldots, w_3)$$

and the lower subpolygon is a degenerated fan, a triangle.

Assume that V_2-V_3 is the only h-arc, then the cost is (see Fig. 2b)

$$w_1 w_2 w_3 + w_2(w_c w_d + w_d w_e + w_e w_f + w_f w_g + w_g w_h + w_h w_i + w_i w_3)$$

$$= T_{123} + w_2(w_c : w_3) \quad , \qquad (1)$$

where w_c:w_3 is the shorthand notation of the sum of adjacent products from w_c to w_3 in the clockwise direction.

Note that the cost of H_0 of the polygon shown in Fig. 2 is

$$\mathrm{Fan}(w_1 | w_2, \ldots, w_3)$$

$$= w_1 (w_2 : w_3) \tag{2}$$

The condition of (1) to be less than (2) is

$$\frac{w_2 \cdot (w_c : w_3)}{(w_2 : w_3) - w_2 \cdot w_3} < w_1$$

Similarly, the condition for the partition in Fig. 2a to be less than H_0 is

$$\frac{w_c \cdot (w_d : w_i)}{(w_c : w_i) - w_c \cdot w_i} < w_1 \quad . \tag{3}$$

We say that a partition is said to be ℓ-optimal among the partitions in a certain class (or several classes) if it is the lexicographically smallest partition among all the partitions with minimum cost in that class (or several classes). Hence, the ℓ-optimum partition is ℓ-optimal among all partitions in the classes $H_0, H_1, \ldots,$ and H_{n-3}.

Now, assume that the ℓ-optimal partition among all the partitions in $H_1, H_2, \ldots, H_{n-3}$ contains only one potential h-arc V_i-V_k only, as shown in Fig. 3. (Note that V_i-V_k will exist in this partition as an h-arc.) This partition will be the ℓ-optimum partition of the monotone basic n-gon if it costs less than that of the fan in H_0. The condition that the partition with V_i-V_k as the single h-arc costs less than H_0 is

$$\frac{w_i \cdot (w_j : w_k)}{(w_i : w_k) - w_i \cdot w_k} < w_1 \quad \text{if } w_i \le w_k$$

or

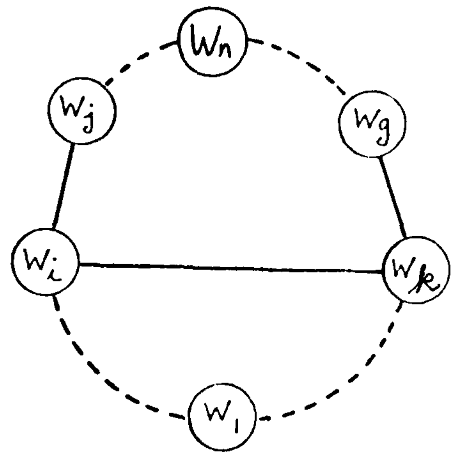


Fig. 3. A monotone polygon with a single h-arc.

$$\frac{w_k \cdot (w_i : w_g)}{(w_i : w_k) - w_i \cdot w_k} < w_1 \quad \text{if } w_k < w_i$$

Combining the two inequalities above, we have

$$\frac{C(w_i, \ldots, w_k)}{(w_i : w_k) - w_i \cdot w_k} < w_1 \tag{4}$$

where $C(w_i, \ldots, w_k)$ denotes the cost of the optimum partition of the subpolygon w_i-w_j-$\cdots$-w_g-w_k and is equal to the cost of the fan in this case.

An h-arc V_i-V_k which divides a polygon into two subpolygons is called a positive arc with respect to the polygon if (4) is satisfied, i.e., the partition with the arc as the only h-arc and a fan in each of the two subpolygons costs less than the fan in the same polygon. Otherwise, it is called a negative arc with respect to the polygon.

When an n-gon is divided into subpolygons, an h-arc is defined as positive in a subpolygon if the cost of partition of the subpolygon with the h-arc as the only h-arc is less than the fan in the subpolygon.

Let us consider a partition with two h-arcs as shown in Fig. 4, and assume that this partition is ℓ-optimal among all partitions in the classes $H_2, H_3, \ldots, H_{n-3}$.

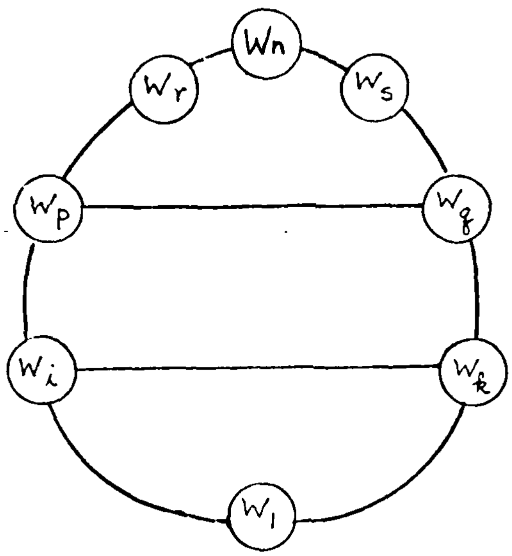


Fig. 4. A monotone 8-gon with two h-arcs.

If V_i-V_k is positive with respect to the subpolygon V_1-V_i-V_p-V_q-V_k, then the condition analogous to (4) is

$$\frac{C(w_i, w_p, w_q, w_k)}{\{(w_i:w_k) - [(w_p:w_q) - w_p \cdot w_q]\} - w_i \cdot w_k} < w_1 \qquad (5a)$$

If V_i-V_k is positive with respect to the whole polygon $V_1 - V_i - \cdots - V_n - \cdots - V_k$, then the condition is

$$\frac{C(w_i, w_p, w_r, w_n, w_s, w_q, w_k)}{(w_i : w_k) - w_i \cdot w_k} < w_l \qquad (5b)$$

Note that (5b) implies (5a).

The condition for the arc $V_p - V_q$ to be positive with respect to the subpolygon $V_i - V_p - V_r - V_n - V_s - V_q - V_k$ is

$$\frac{C(w_p, w_r, w_n, w_s, w_q)}{(w_p : w_q) - w_p \cdot w_q} < \min(w_i, w_k) \quad . \qquad (6a)$$

If the arc $V_p - V_q$ is positive with respect to the whole polygon $V_l - V_i - V_p - V_r - V_n - V_s - V_q - V_k$, it must satisfy (6b).

$$\frac{C(w_p, w_r, w_n, w_s, w_q)}{(w_p : w_q) - w_p \cdot w_q} < w_l \quad . \qquad (6b)$$

Since $w_l < \min(w_i, w_k)$, condition (6b) implies (6a).

Here, the presence of $V_i - V_k$ will divide the original polygon into two subpolygons where $V_p - V_q$ appears in the upper subpolygon. If $V_p - V_q$ is a positive arc with respect to the original polygon, then $V_p - V_q$ is certainly positive in the upper subpolygon. But if $V_p - V_q$ is positive in the subpolygon, the arc $V_p - V_q$ may become negative if $V_i - V_k$ is removed, i.e. $V_p - V_q$ becomes negative with respect to the original polygon.

Similarly, if the arc $V_i - V_k$ is positive with respect to a subpolygon, the arc $V_i - V_k$ may become negative if the arc $V_p - V_q$ is removed.

The preceding discussions can be summarized as Theorem 1.

Theorem 1. If an h-arc is positive with respect to a polygon then the arc is positive with respect to any subpolygon containing that arc. If an h-arc is positive with respect to a subpolygon, it may or may not be positive with respect to a larger polygon which contains the subpolygon. ■

There are two intuitive approaches to the ℓ-optimum partition of a monotone basic polygon. The first approach is to put in the potential h-arcs one by one. Each additional potential h-arc will improve the cost until the correct number of h-arcs is reached. Any further increase in the number of h-arcs will increase the cost. To introduce an h-arc into the polygon, we can test each potential h-arc (at most n-3) to see if it is positive with respect to the whole polygon. If yes, that positive arc must exist in the ℓ-optimum partition, and the polygon will be divided into two subpolygons, each being a monotone polygon. We can repeat the whole process of testing positiveness of the h-arcs. The trouble is that all these arcs may be negative individually with respect to the whole polygon and yet H_0 may not be the optimum. For example, two arcs V_i-V_j and V_p-V_q may be negative individually with respect to the whole polygon but the partition with both V_i-V_j, V_p-V_q present at the same time may cost less than H_0 as shown in Fig. 5a. This shows that we cannot guarantee an optimum partition simply because no more potential h-arcs can be added one at a time.

The second approach is to put all the potential h-arcs in first and then take out the potential h-arcs one-by-one, where each deletion

will decrease the cost until the correct number of h-arcs is reached. Any further deletions will increase the cost. Unfortunately, even if all h-arcs are positive with respect to their subpolygon, the partition may not be optimum. In Fig. 5b, each h-arc is positive with respect to its local subpolygon but the partition is not optimum. (Note that positiveness of an h-arc in a quadrilateral is the same as stability. But the idea of stability applied to vertical arcs as well.) This means that we cannot guarantee an optimum partition simply because no h-arc can be deleted one at a time.

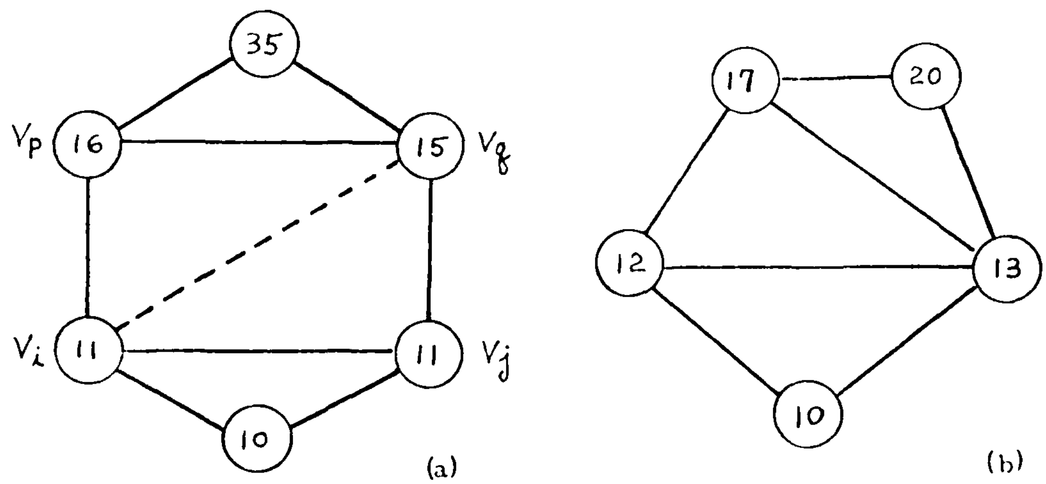


Fig. 5. Counter examples for the intuitive approaches.

Let us outline the idea of an O(n) algorithm for finding the ℓ-optimum partition of a monotone basic polygon. First, we get all the potential h-arcs by the one-sweep algorithm. Then, we start from the highest potential h-arc and process each potential h-arc from the highest to the lowest. For each potential h-arc, we try to get the ℓ-optimum partition of the upper subpolygon of that arc (i.e. the ℓ-optimum partition of the subpolygon bounded below by that h-arc). The ℓ-optimum partition in the subpolygon is obtained by comparing the cost of the ℓ-optimal partition among the partitions of the upper subpolygon which contain one or more potential h-arcs with that of the fan in the upper subpolygon.

If we try all possible combinations of the potential h-arcs as candidates for the ℓ-optimal partitions, we need $O(n^3)$ operations to find the ℓ-optimum partition. Fortunately, there are some dependence-relationships among these potential h-arcs. Hence, certain subsets of the potential h-arcs will either all exist or all disappear in the ℓ-optimum partition of the monotone polygon. We shall be dealing with potential h-arcs most of the time, so we shall use "arcs" instead of potential h-arcs for brevity.

Consider the monotone basic polygon shown symbolically in Fig. 6. There are three potential h-arcs, denoted by h_k, h_j, and h_i. V_n is the local maximum vertex and V_1 is the local minimum vertex. Without loss of generality, we can assume $w_a \le w_a'$ for $a = i$, j and k. Since we shall deal with subpolygons bounded by two potential h-arcs, let us use h_n for V_n and h_1 for V_1 (i.e. we consider these vertices as

degenerated arcs). From Lemmas 1 and 3, the ℓ-optimum partitions of the subpolygons bounded by two potential h-arcs (i.e. the white area of the polygon in Fig. 6) are all fans.

Assume (i) h_k is positive in the subpolygon bounded by h_n and h_j but h_k is negative in the subpolygon bounded by h_n and h_i,

(ii) h_j is positive in the subpolygon bounded by h_k and h_i but h_j is negative in the subpolygon bounded by h_k and h_1, and

(iii) h_i is positive in the subpolygon bounded by h_j and h_1 only.

Then either the three arcs h_k, h_j, h_i all exist or no h-arcs exists in the optimum partition.

This shows that the existence of an h-arc depends on the existence of another h-arc.

We shall use the notations

$C\begin{pmatrix} h_j \\ h_i \end{pmatrix}$ to denote the cost of the ℓ-optimum partition of the subpolygon bounded above by h_j and bounded below by h_i, and

$H_0\begin{pmatrix} h_j \\ h_i \end{pmatrix}$ to denote the cost of the fan in the subpolygon bounded above by h_j and bounded below by h_i.

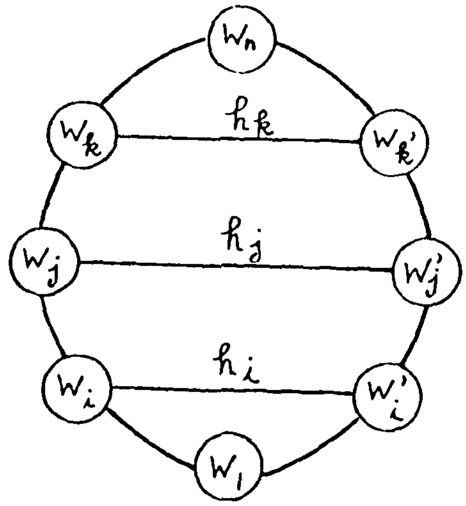


Fig. 6. An octagon with three potential h-arcs.

In Fig. 6, the condition for h_k to be positive with respect to the whole polygon is (compare (5a))

$$\frac{C\begin{pmatrix} h_n \\ h_k \end{pmatrix}}{(w_k : w_k') - w_k \cdot w_k'} < w_1 \tag{7}$$

The LHS of (7) is denoted by

$$S\begin{pmatrix} h_n \\ h_k \end{pmatrix}$$

and is called the <u>supporting weight</u> of h_k with h_n as the ceiling (the definition of ceiling will be given formally later). Note that the LHS of (7) depends only on the weights of vertices in the upper subpolygon of h_k.

In terms of the supporting weights, we can write the three conditions (i), (ii) and (iii) as follows:

(i) $$w_i < S\begin{pmatrix} h_n \\ h_k \end{pmatrix} < w_j$$

(ii) $$w_1 < S\begin{pmatrix} h_k \\ h_j \end{pmatrix} < w_i$$

(iii) $$S\begin{pmatrix} h_j \\ h_k \end{pmatrix} < w_1 \ .$$

An arc h_j is a <u>son</u> of the arc h_i (or h_i is the <u>father</u> of h_j) if the following conditions are satisfied:

(i) h_j is above h_i (the son is above the father)

(ii) In any subpolygon containing h_i and h_j, the arc h_j will exist in the ℓ-optimum partition of the subpolygon if and only if h_i exists in the ℓ-optimum partition.

(iii) h_i is the highest arc that satisfies (i) and (ii).

It is easy to see that every arc can have at most one father but an arc can have many sons. Also the ancestor-descendant relationship is a transitive relationship. If an arc exists in the ℓ-optimum partition, all its descendants will also exist.

An arc h_k is a <u>ceiling</u> of an arc h_i if the following conditions are satisfied:

(i) h_k is above h_i

(ii) h_k is <u>not</u> a descendant of h_i

(iii) h_k is the lowest arc which satisfies (i) and (ii).

Consider two partitions of a subpolygon as shown in Fig. 7.

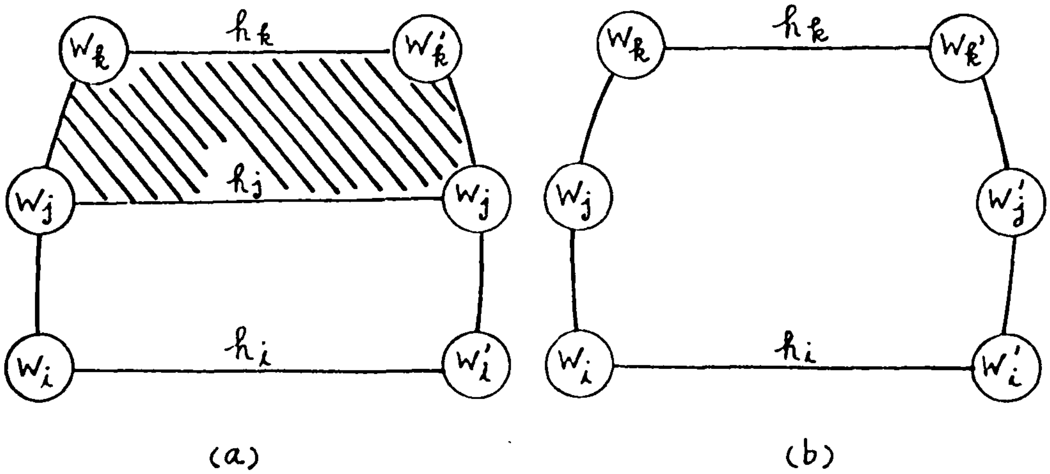


Fig. 7. A subpolygon of the octagon shown in Fig. 6 (The shaded areas are optimally partitioned and the blank areas are partitioned by a fan. The h-arcs in the shaded area are all descendants of h_j.)

The cost of partition of Fig. 7a is

$$C\begin{pmatrix} h_k \\ h_j \end{pmatrix} + H_0\begin{pmatrix} h_j \\ h_i \end{pmatrix}$$

where the cost of partition in Fig. 7b is

$$H_0\begin{pmatrix} h_k \\ h_i \end{pmatrix}$$

The condition for the partition in Fig. 7a to be cheaper than that in Fig. 7b is (similar to (5a))

$$S\begin{pmatrix} h_k \\ h_j \end{pmatrix} < w_i \quad .$$

In order to give an intuitive meaning of the supporting weight $S\begin{pmatrix} h_k \\ h_j \end{pmatrix}$, let us regard h_k and h_j in Fig. 7 as fixed while the position of h_i can be moved up or down by increasing or decreasing the values of w_i and w_i'. If h_i moves up and coincides with h_j, i.e., $w_i = w_j$, the partition in Fig. 7a costs less than or equal to the partition in Fig. 7b. If the position of h_i moves down gradually from h_j, there will be a position for which the cost of the partition in Fig. 7a is equal to the cost of the partition in Fig. 7b. We can consider this position as a <u>fictitious</u> arc f_j, i.e.

$$C\begin{pmatrix} h_k \\ h_j \end{pmatrix} + H_0\begin{pmatrix} h_j \\ f_j \end{pmatrix} = H_0\begin{pmatrix} h_k \\ f_j \end{pmatrix} \quad , \tag{8}$$

the ℓ-optimum partition of the subpolygon bounded by h_k and h_i becomes a fan. The arc f_j is called the <u>floor</u> of h_j. Note that the minimum of the two weights associated with f_j is the supporting weight of h_j.

We now give two examples to illustrate the concepts, notations and the algorithms. Then a formal description of the algorithm will be given.

Consider a monotone basic polygon with five potential h-arcs, $h_6, h_5, \ldots, h_2$ where h_6 is the highest arc as shown symbolically in Fig. 8. Let $w_i < w_i'$ for $i = a, b, \ldots, e$. The maximum vertex, which lies above h_6, has the weight w_f and the minimum vertex, which lies below h_2, has the weight w_1. We can regard w_f (and w_1) as a degenerated arc and use h_7 to represent w_f (and h_1 to represent w_1).

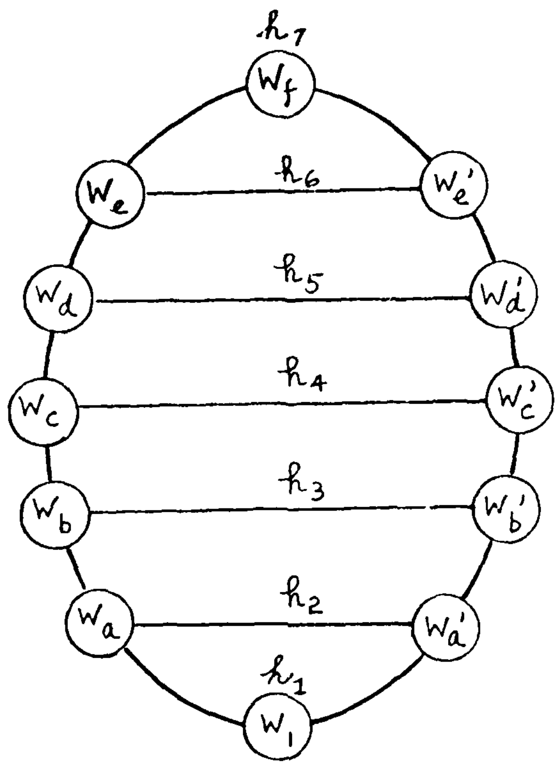


Fig. 8. A 12-gon with 5 h-arcs.

Example 1

Let us write down the comparisons made in the algorithms.

First, we compare

$$H_0\binom{h_7}{h_6} + H_0\binom{h_6}{f_6} = H_0\binom{h_7}{f_6}$$

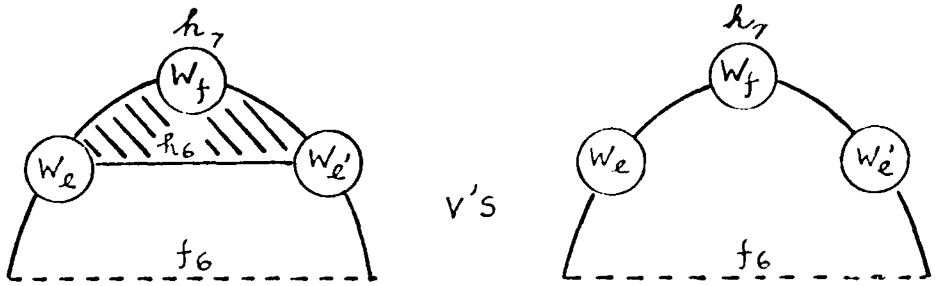


Fig. 9. Illustrations for Example 1.
9a. To find f_6.

In the equation, f_6 is the only unknown. In computation, we do not use the equation but use the supporting weight of h_6 instead (h_7 is the ceiling of h_6). If the h-arc h_5 is below or coincides with f_6, which means that h_6 is negative with respect to the smallest subpolygon, h_6 should be deleted and never appear in the f-optimum partition. For *simplicity, we shall assume all arcs and floors have distinct positions in the example.*

Let us assume that f_6 is below h_5, or symbolically we write

$$h_5/f_6 \quad .$$

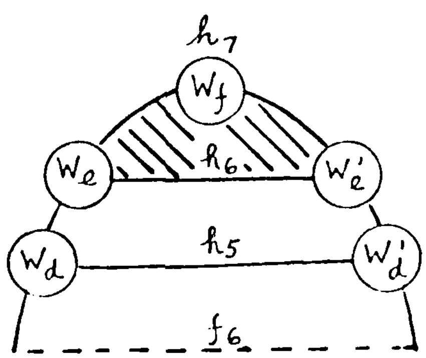


Fig. 9b. The position of f_6.

Then we do the next comparison.

$$H_0\binom{h_6}{h_5} + H_0\binom{h_5}{f_5} = H_0\binom{h_6}{f_5}$$

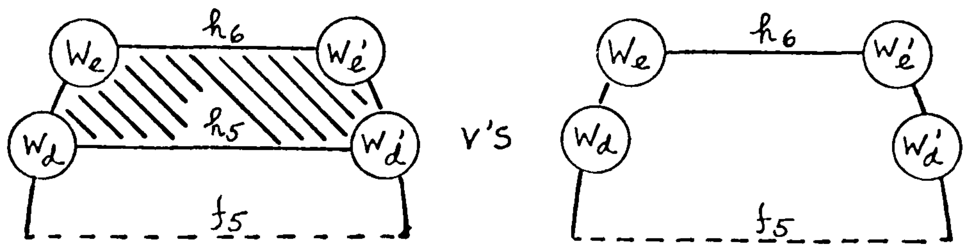


Fig. 9c. To find f_5.

Assume that f_6/f_5, i.e. h_6 is a son of h_5, and h_4/f_5, the next comparison is

$$H_0\binom{h_7}{h_6} + H_0\binom{h_6}{h_5} + H_0\binom{h_5}{f_{65}} = H_0\binom{h_7}{f_{65}} .$$

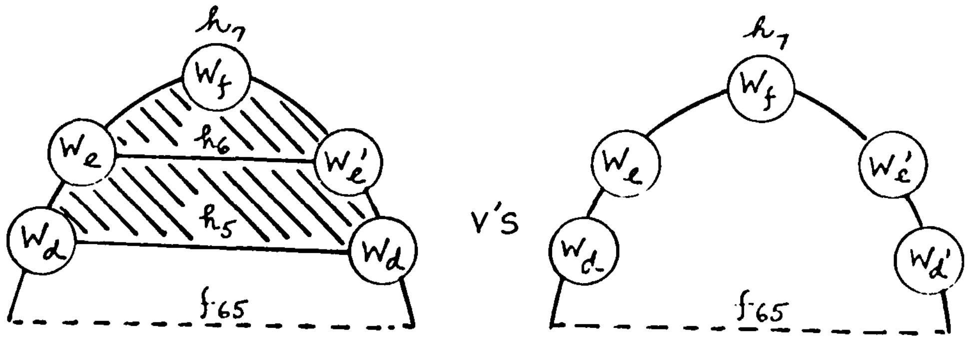


Fig. 9d. Condense h_6 to h_5 and find f_{65}.

Note that f_{65} is in a sense the combined floor of h_6 and h_5 and h_7 becomes the ceiling of h_5. The equation can also be written as

$$C\binom{h_7}{h_5} + H_0\binom{h_5}{f_{65}} = H_0\binom{h_7}{f_{65}}$$

If h_4/f_{65}, the next comparison will be

$$H_0\binom{h_5}{h_4} + H_0\binom{h_4}{f_4} = H_0\binom{h_5}{f_4}$$

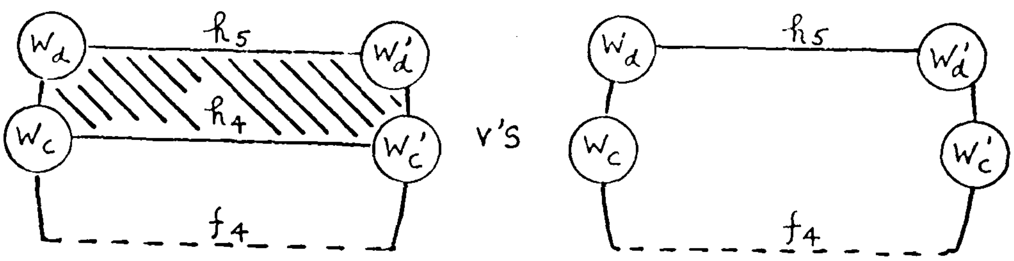


Fig. 9c. To find f_4.

Assume that f_{65}/f_4, i.e. h_5 is a son of h_4, and h_3/f_4, we have

$$C\binom{h_7}{h_4} + H_0\binom{h_4}{f_{654}} = H_0\binom{h_7}{f_{654}}$$

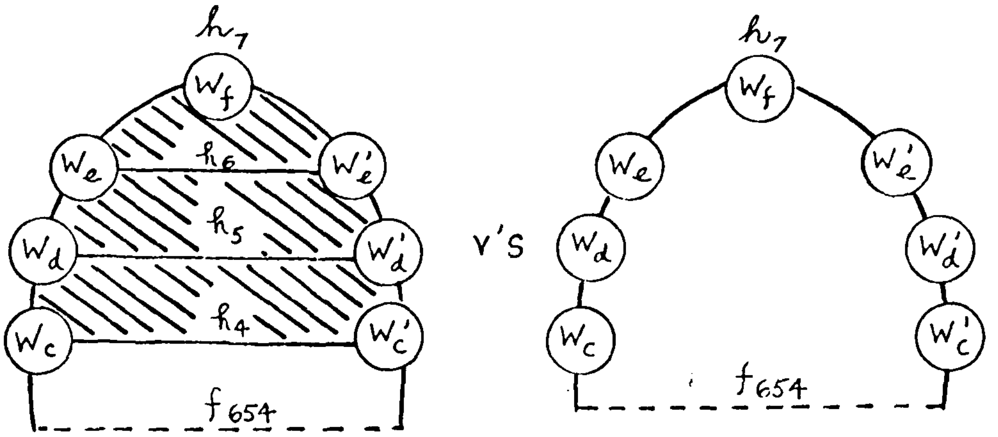


Fig. 9f. To find f_{654}.

with h_7 as the ceiling of h_4. Moving to h_3, we compare

$$H_0\binom{h_4}{h_3} + H_0\binom{h_3}{f_3} = H_0\binom{h_4}{f_3}$$

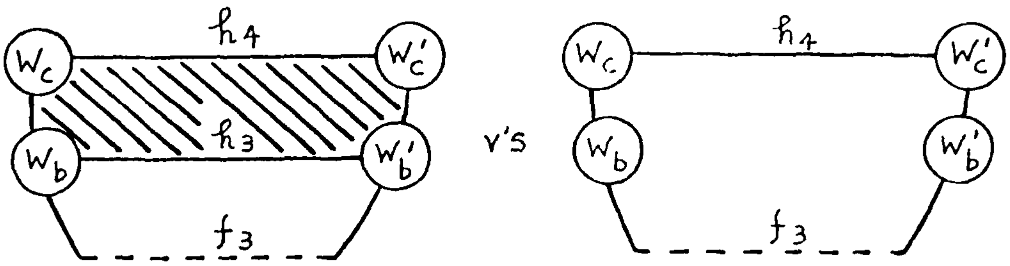


Fig. 9g. To find f_3.

Assume that f_{654}/f_3, i.e. h_4 is a son of h_3 and h_2/f_3, we compare

$$C\begin{pmatrix} h_7 \\ h_3 \end{pmatrix} + H_0\begin{pmatrix} h_3 \\ f_{6543} \end{pmatrix} = H_0\begin{pmatrix} h_7 \\ f_{6543} \end{pmatrix}$$

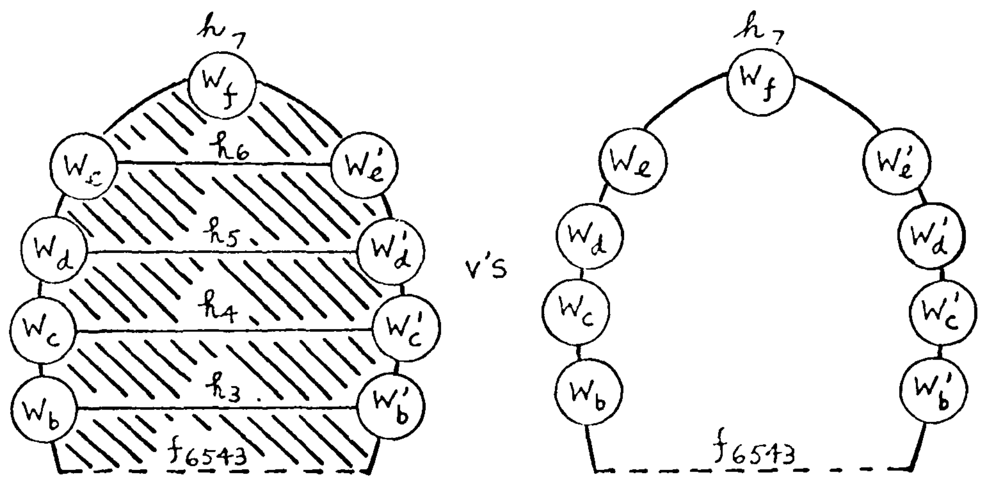


Fig. 9h. To find f_{6543}.

with h_7 as the ceiling of h_3. Moving to h_2, we compare

$$H_0\begin{pmatrix} h_3 \\ h_2 \end{pmatrix} + H_0\begin{pmatrix} h_2 \\ f_2 \end{pmatrix} = H_0\begin{pmatrix} h_3 \\ f_2 \end{pmatrix}$$

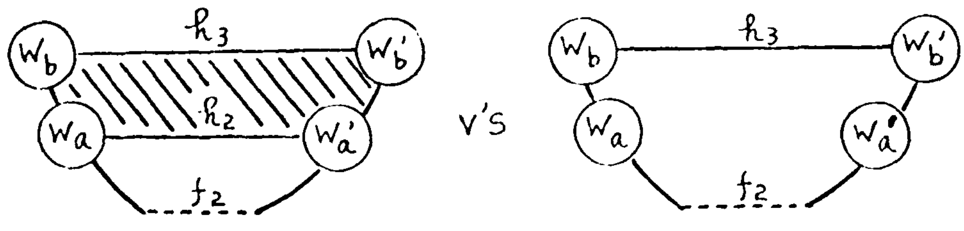


Fig. 9i. To find f_2.

Assume that f_{6543}/f_2, i.e. h_3 is a son of h_2, and h_1/f_2, we have

$$C\binom{h_7}{h_2} + H_0\binom{h_2}{f_{65432}} = H_0\binom{h_7}{f_{65432}}$$

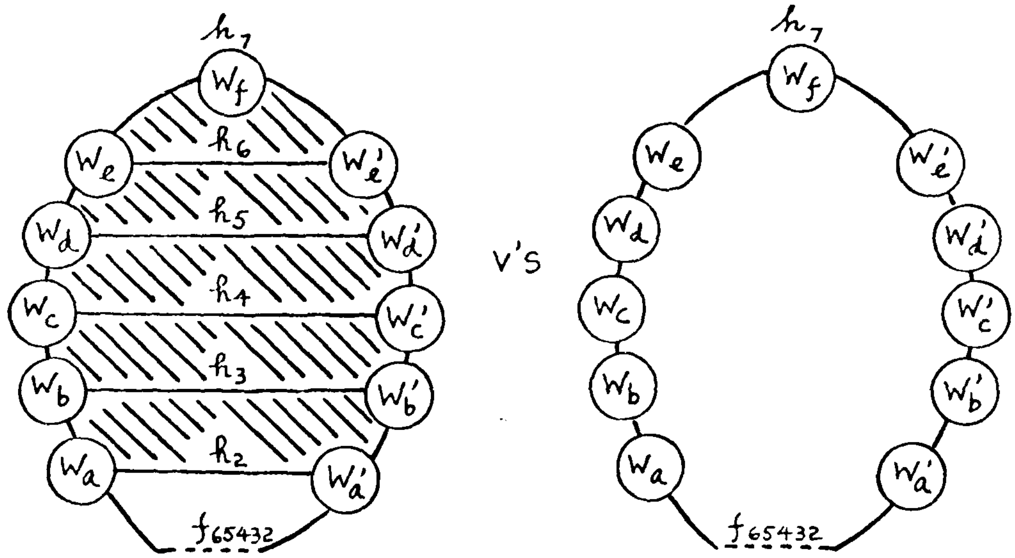


Fig. 9j. To find f_{65432}.

and h_7 is the ceiling of h_2. Now if

$$S\binom{h_7}{h_2} < w_1 ,$$

the partition consisting of

$$C\binom{h_7}{h_2} + H_0\binom{h_2}{h_1}$$

is the ℓ-optimum partition.

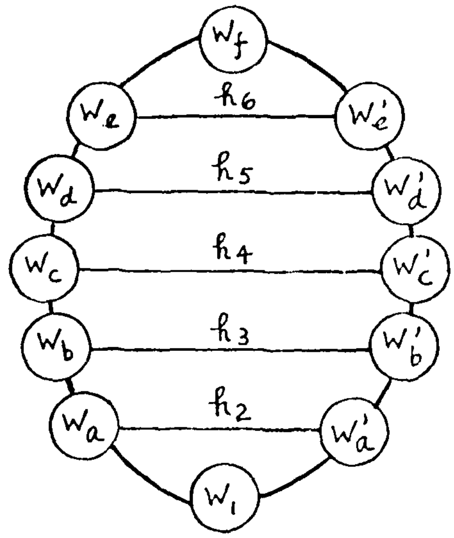


Fig. 9k. The ℓ-optimum partition.

If $S\begin{pmatrix} h_7 \\ h_2 \end{pmatrix} \geq w_1$, then $H_0\begin{pmatrix} h_7 \\ h_1 \end{pmatrix}$ will be the ℓ-optimum partition.

<u>Example 2.</u> The successive comparisons are

$$H_0\begin{pmatrix} h_7 \\ h_6 \end{pmatrix} + H_0\begin{pmatrix} h_6 \\ f_6 \end{pmatrix} = H_0\begin{pmatrix} h_7 \\ f_6 \end{pmatrix}$$

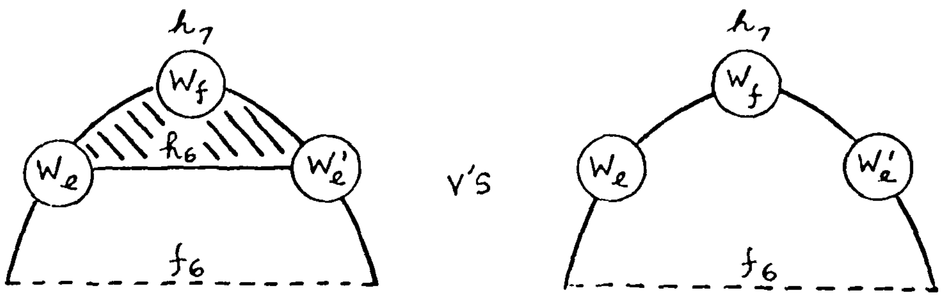


Fig. 10. Illustrations for Example 2.
10a. To find f_6.

Assume that h_5/f_6, we compare

$$H_0\begin{pmatrix} h_6 \\ h_5 \end{pmatrix} + H_0\begin{pmatrix} h_5 \\ f_5 \end{pmatrix} = H_0\begin{pmatrix} h_6 \\ f_5 \end{pmatrix}$$

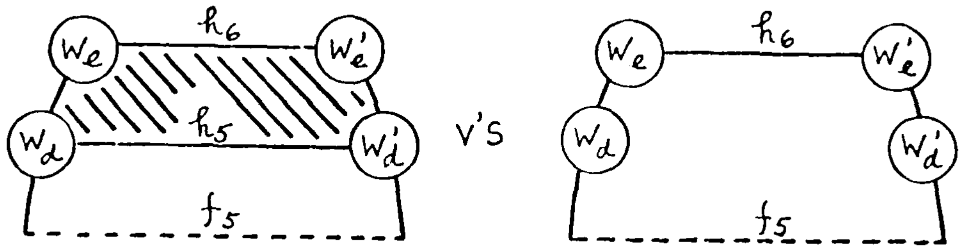


Fig. 10b. To find f_5.

Assume that f_5/f_6, i.e. h_6 becomes the ceiling of h_5, and h_4/f_5, we compare

$$H_0\begin{pmatrix} h_5 \\ h_4 \end{pmatrix} + H_0\begin{pmatrix} h_4 \\ f_4 \end{pmatrix} = H_0\begin{pmatrix} h_5 \\ f_4 \end{pmatrix}$$

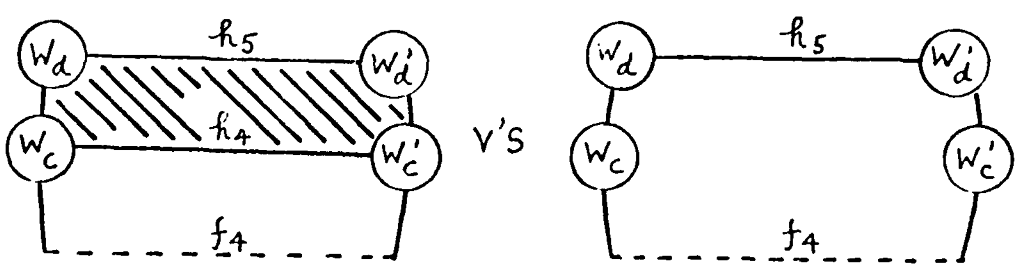


Fig. 10c. To find f_4.

Assuming that f_4/f_5, i.e. h_5 becomes the ceiling of h_4, and h_3/f_4, we compare

$$H_0\begin{pmatrix}h_4\\h_3\end{pmatrix} + H_0\begin{pmatrix}h_3\\f_3\end{pmatrix} = H_0\begin{pmatrix}h_4\\f_3\end{pmatrix}$$

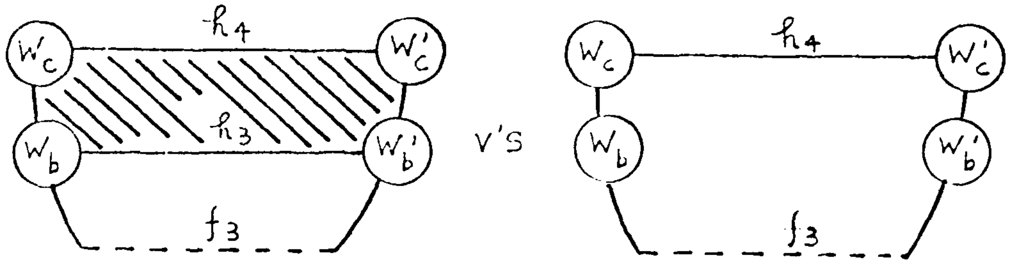


Fig. 10d. To find f_3.

Assume that f_3/f_4 and f_3/h_2, then arc h_3 should be deleted. Next, we assume that f_4/h_2, then arc h_4 should also be deleted. Suppose h_2/f_5, we shall then compare

$$H_0\begin{pmatrix}h_5\\h_2\end{pmatrix} + H_0\begin{pmatrix}h_2\\f_2\end{pmatrix} = H_0\begin{pmatrix}h_5\\f_2\end{pmatrix}$$

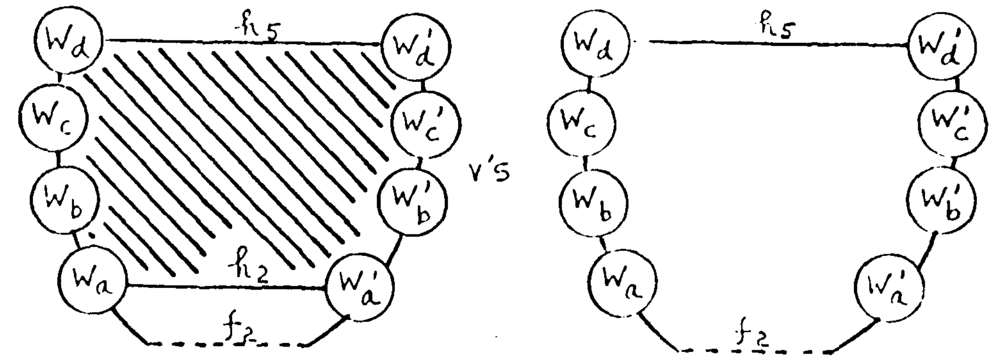


Fig. 10e. To find f_2.

Assume f_5/f_2, i.e. h_5 is a son of h_2 and h_1/f_2, we then determine f_{52},

$$C\begin{pmatrix} h_6 \\ h_2 \end{pmatrix} + H_0\begin{pmatrix} h_2 \\ f_{52} \end{pmatrix} = H_0\begin{pmatrix} h_6 \\ f_{52} \end{pmatrix}$$

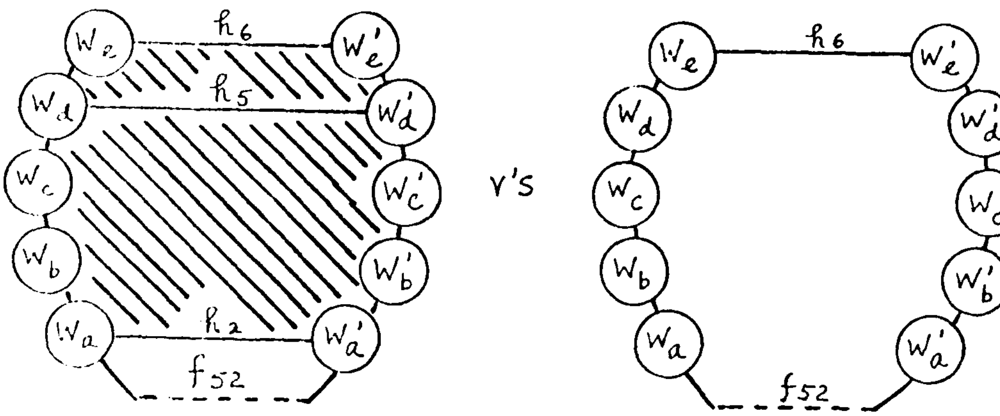


Fig. 10f. To find f_{52}.

Assume f_6/f_{52}, i.e. h_6 is a son of h_2, and h_i/f_{52}, our next comparison is

$$C\begin{pmatrix} h_7 \\ h_2 \end{pmatrix} + H_0\begin{pmatrix} h_2 \\ f_{652} \end{pmatrix} = H_0\begin{pmatrix} h_7 \\ f_{652} \end{pmatrix}$$

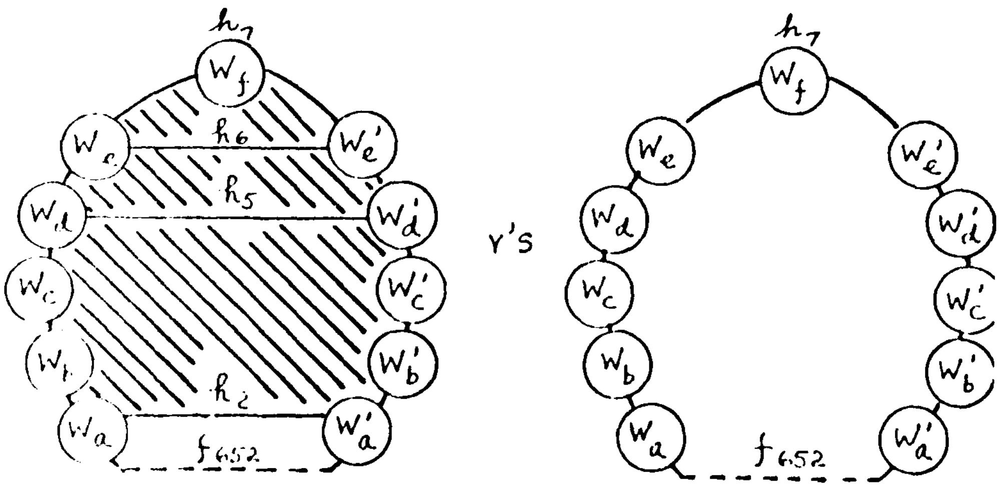


Fig. 10g. To find f_{652}.

and h_7 becomes the ceiling of h_2.

Assume h_1/f_{652}, the the partition $C\binom{h_7}{h_2} + H_0\binom{h_2}{h_1}$ is the ℓ-optimum partition.

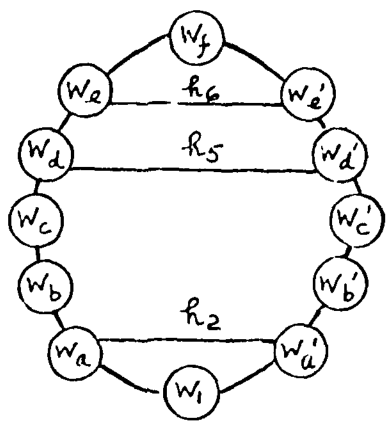


Fig. 10h. The ℓ-optimum partition.

Had we assumed f_{52}/f_6 and f_{52}/h_1 then both h_5 and h_2 should also be removed and we are left with

$$f_6 \text{ against } h_1 .$$

If h_1/f_6, then we have the ℓ-optimum partition

$$H_0\binom{h_7}{h_6} + H_0\binom{h_6}{h_1} .$$

From the above two examples, we can see that h_k is the ceiling of h_i if h_k is the lowest arc above h_i such that the supporting weight of h_k is smaller than or equal to that of h_i.

Let us outline the algorithm for finding the ℓ-optimum partition of a monotone basic polygon.

1. Get all the potential h-arcs of the polygon by the one-sweep algorithm. (All the h-arcs form a list with the arc V_b-$V_{b'}$ at the bottom.)

2. Process the potential h-arcs one by one, from the top to the bottom. (We try to find the ℓ-optimum partition of the subpolygon bounded below by the arc being processed.)

   2a. Let h_R be the arc currently being examined, h_C be the arc immediately above h_R, and h_N be the arc immediately below h_R in the list. If h_R is negative with respect to the subpolygon bounded above by h_C and below by h_N, delete h_R, otherwise go to Step 2c.

   2b. Once h_R and its descendants are deleted, we backtrack to h_C and compare the cost of the partition with h_C and its descendants against the cost of the fan in the subpolygon bounded above by the ceiling of h_C and below by h_N. If the fan is ℓ-optimum in the subpolygon, we will delete h_C and repeat this step until no further deletion is possible. Then we move to examine h_N. (The actual comparisons are done in terms of the supporting weights.)

   2c. Here, h_R is positive in the smallest subpolygon bounded by potential h-arcs. We will backtrack to condense all its descendants to h_R as follows. Let h'_C be the ceiling of h_C. If

$S\begin{pmatrix} h_C \\ h_R \end{pmatrix} < S\begin{pmatrix} h_C' \\ h_C \end{pmatrix}$, h_C becomes a son of h_R. We will combine h_C as well as all its descendants to h_R and recalculate the combined supporting weight $S\begin{pmatrix} h_C' \\ h_R \end{pmatrix}$.

Replace h_C by h_C' and compare the cost of the partition with h_R and its descendants against that of the fan in the sub-polygon bounded above by the new h_C, i.e. h_C', and below by h_N. If the fan is ℓ-optimum in the subpolygon, we delete h_R as well as its descendants, and go to Step 2b to see if we can delete more arcs. Otherwise, we repeat this step to see if we can condense more arcs.

2d. Now we have $S\begin{pmatrix} h_C \\ h_R \end{pmatrix} \geq S\begin{pmatrix} h_C' \\ h_C \end{pmatrix}$, the supporting weight of h_C. The arc h_C is the ceiling of h_R and $S\begin{pmatrix} h_C \\ h_R \end{pmatrix}$ is the supporting weight of h_R. We move and process h_N.

Before a formal description of the algorithm is given, a procedure to process the list of potential h-arcs in a monotone polygon is presented.

Procedure MONO-PARTITION (L)

Input: consists of a list of potential h-arcs, passed to the procedure via the argument L. Let h_1 be the lowest arc in L, the one immediately above h_1 be h_2, and h_{p+1} be the highest arc in L. (Note that h_1 and h_{p+1} are degenerated arcs with the minimum vertex and the maximum vertex of the polygon.)

Output: consists of all the potential h-arcs that exist in the ℓ-optimum partition of the polygon.

Step 0 $h_C := h_{p+1}$;

$h_R := h_p$;

$h_N := h_{p-1}$;

MIN-WEIGHT := minimum of the two vertices of h_N ;

Comment: h_R is the arc to be processed and h_C is the ceiling of the subpolygon. h_N is the arc immediately below h_R in L.

Step 1 Calculate $S\binom{h_C}{h_R}$;

If $S\binom{h_C}{h_R} \geq$ MIN-WEIGHT

then go to Step 2

else go to Step 3 .

Step 2 While ($h_R \neq h_{p+1}$) And

(the supporting weight of $h_R \geq$ MIN-WEIGHT) Do

Begin

Remove h_R and all its descendants from L;

$h_R := h_C$;

$h_C :=$ the ceiling of the new h_R

End;

Go to Step 4.

Step 3 If ($h_C \neq h_{p+1}$) and (the supporting weight of $h_R <$ the supporting weight of h_C)

then

Begin

Condense h_C and all its descendants into h_R ;

$h_C :=$ the ceiling of h_C;

go to Step 1;

End

else

Begin

Record $S\begin{pmatrix} h_C \\ h_R \end{pmatrix}$ as the supporting weight of h_R and h_C as the ceiling of h_R;

go to Step 4;

End.

Step 4   If $h_N \neq h_1$

then

Begin

$h_C := h_R$;

$h_R := h_N$;

$h_N :=$ the arc immediately below the new h_R;

MIN-WEIGHT := minimum of the two vertices of the new h_N;

go to Step 1;

End

else go to Step 5 ;

Step 5   Exit procedure and return L to caller.

Now we can give the algorithm for finding the ℓ-optimum partition of a monotone basic polygon.

Algorithm I

Input   consists of n positive integers, which are the weights of the n vertices of the monotone n-gon. W[1] is the weight of the minimum vertex and W[i+1] is the neighbor of W[i] of the n-gon going in the clockwise direction. Let the weight of the maximum vertex be W[t].

Output   consists of a list of potential h-arcs which will exist in the ℓ-optimum partition of the n-gon, the partitions in the sub-polygons bounded by every two consecutive arcs in the list are fans.

Step 0 For $i := 2$ step 1 until N do

$$CP[i] := \sum_{j=1}^{i-1} W[j] \cdot W[j+1];$$

$CP[1] := 0;$

Comment: The sum of adjacent products $W[i]; W[j]$ can be obtained from $CP[j] - CP[i]$ for $1 < i < j \leq N$ and hence we can calculate the supporting weights easily.

Step 1 Apply the one-sweep algorithm to obtain a list of arcs. Let this list be L.

Comment: L contains (n-3) arcs which includes all potential h-arcs in the monotone n-gon, and these arcs are layered, one above another.

Step 2 From L, remove those arcs which are not potential h-arcs;

If L is empty

then go to Step 6

else go to Step 3.

Step 3 Let the lowest arc in L be h_2, the one immediately above h_2 be h_3, and so on:

Let the highest arc in L be h_p;

Insert h_1 with weight $W[1]$ below h_2;

Insert h_{p+1} with weight $W[t]$ above h_p.

Comment: h_{p+1} is the ceiling of h_p.

Step 4 MONO-PARTITION (L);

Comment: when returned from MONO-PARTITION, L will contain all the ceiling arcs with their descendants in the ℓ-optimum partition.

Step 5 Remove h_1 and h_{p+1} from L;

Step 6 Output L and stop.

This algorithm has been implemented in Pascal and the listing of the computer program is given in Appendix I.

Lemma 5. Any arc which is deleted from the arc-list L in Step 2 of the procedure MONO-PARTITION cannot be present in the ℓ-optimum partition of the polygon.

Proof. There are two cases in which an arc is deleted from L:

(1) Its ancestors are deleted. It follows from the definition of the ancestor-descendant relationship that it cannot be present in the ℓ-optimum partition of the polygon.

(2) It is the h_R which satisfies the logical condition of Step 2 of the procedure. Hence, in the subpolygon bounded below by h_N and above by h_C, the partition with h_R and its descendants costs more than or equal to that of the fan. Hence, the partition with h_R and its descendants is not ℓ-optimum in the subpolygon and h_R as well as its descendants should not appear in the ℓ-optimum partition of the whole polygon. ■

Lemma 6. After an arc h_i has been processed, the subpolygon between h_i and its ceiling is optimally partitioned.

Proof. The h-arcs remaining in the partition of the subpolygon are all descendants of h_i. By definition of the ancestor-descendant relationship, the partition of the subpolygon is optimum. ■

Lemma 7. Let V_t be the maximum vertex, and $h_k, h_{k-1}, \ldots, h_{j+1}$ be a set of h-arcs in the partition such that

$$h_k / h_{k-1} / \ldots / h_{j+1} / h_j \quad ,$$

and

$$h_k \text{ is the ceiling of } h_{k-1} \quad ,$$
$$\vdots$$
$$h_{j+1} \text{ is the ceiling of } h_j \quad ,$$

then the supporting weights of these h-arcs satisfy

$$S\begin{pmatrix} h_t \\ h_k \end{pmatrix} \leq S\begin{pmatrix} h_k \\ h_{k-1} \end{pmatrix} \leq \cdots \leq S\begin{pmatrix} h_{j+1} \\ h_j \end{pmatrix} \quad . \tag{9}$$

Proof. Assume that one of the inequalities is not satisfied, say

$$S\begin{pmatrix} h_{j+2} \\ h_{j+1} \end{pmatrix} > S\begin{pmatrix} h_{j+1} \\ h_j \end{pmatrix} \quad .$$

Then if h_j exists h_{j+1} will also exist, h_{j+1} becomes a son of h_j. This contradicts the assumption that h_{j+1} is a ceiling of h_j. ■

Lemma 8. Any arc which remains in L at the end of the procedure must be present in the ℓ-optimum partition of the polygon.

Proof. We can divide the h-arcs in L at the end of the procedure into two groups:

(i) those which are descendant of some other arcs in the output, and

(ii) those which have no ancestor in the output.

By the definition of the descendant-ancestor relationship, the arcs in group (i) must be present in the ℓ-optimum partition whenever their corresponding ancestors in group (ii) is present in the ℓ-optimum partition. Hence, we have only to show that all arcs in group (ii) must be present in the ℓ-optimum partition.

Let V_t be the maximum vertex and the set of arcs in group (ii) be $h_k, h_{k-1}, \ldots, h_{j+1}, h_j$ such that $h_k/h_{k-1}/\cdots/h_{j+1}/h_j$. Since none of these arcs has an ancestor, we must have

h_k as the ceiling of h_{k-1},

$\vdots$

and h_{j+1} as the ceiling of h_j.

From the logical condition in Step 1 of the procedure, we have

$$w_1 > S\binom{h_{j+1}}{h_j} \tag{10}$$

From Lemma 7 and (10), we have

$$w_1 > S\binom{h_{j+1}}{h_j} > \cdots > S\binom{h_k}{h_{k-1}} > S\binom{h_t}{h_k},$$

which implies that

$$H_0\binom{h_t}{h_1} > \left[C\binom{h_t}{h_k} + H_0\binom{h_k}{h_1}\right]$$

$$> \left[C\binom{h_t}{h_k} + C\binom{h_k}{h_{k-1}} + H_0\binom{h_{k-1}}{k_1}\right]$$

$$\vdots$$

$$> \left[C\binom{h_t}{h_k} + C\binom{h_k}{h_{k-1}} + \cdots + C\binom{h_{j+1}}{h_j} + H_0\binom{h_j}{h_1}\right]$$

= the cost of the ℓ-optimum partition of the polygon.

In other words, for any arc h_i in group (ii) of L, $i = k, k-1, \ldots, j+1, j$, all the arcs above h_i in L must be present in ℓ-optimum partition of the upper subpolygon of h_i. Since $H_0\binom{h_t}{h_1} > C\binom{h_t}{h_j} + H_0\binom{h_j}{h_1}$, they all should be present in the ℓ-optimum partition of the monotone basic polygon. ■

Theorem 2. The partition obtained by the algorithm is ℓ-optimum.

Proof. From Theorems 3 and 4 of Part I, we know that all the h-arcs present in the ℓ-optimum partition are potential h-arcs and hence are included in the arc-list L obtained by the one-sweep algorithm. It follows from Lemmas 5 and 8 that all the arcs which are deleted from L cannot be present in the ℓ-optimum partition and all the arcs which remain in L must be present in the ℓ-optimum partition. Further, from Lemma 1, the

ℓ-optimum partition in any subpolygon bounded by two adjacent potential h-arcs in L must be a fan. Hence, the partition consisting of the h-arcs output by the algorithm and with fans in every subpolygons bounded by two adjacent arcs in L must be ℓ-optimum. ■

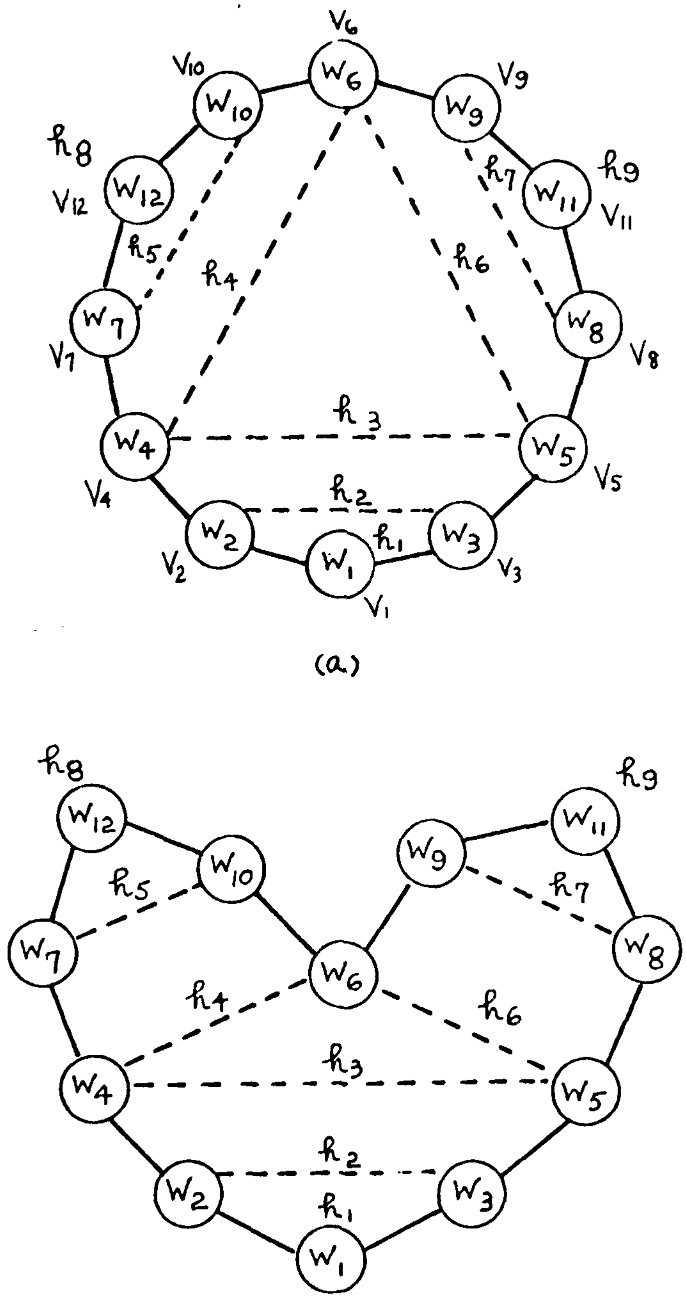
Let us examine how much time we spend in executing the algorithm.

Step 0 and Step 1 each scans the polygon once, and hence takes O(n) time. Since there are at most n-3 arcs in L, Step 2 also takes O(n) time. There are three nested loops in the procedure. The innermost one is in Step 6, the middle one spans from Step 1 to Step 3, while the outermost one spans from Step 1 to Step 5. Whenever the innermost loop is executed once, a potential h-arc is deleted from L. Whenever the middle loop is executed once (i.e. the "then" part of Step 3 is executed once), a potential h-arc is condensed into its father. Once an arc is deleted or condensed, it will never be examined again. Since there are at most n-3 potential h-arcs in L, the total number of executions in Step 2 and Step 3 is O(n). The outermost loop will also be executed at most (n-3) times. Hence the whole algorithm will finish its work in O(n) time.

3. The Convex Polygon

There may be several local maximum vertices in a general convex polygon. Let us still draw the polygon in such a way that the global minimum vertex is at the bottom. From Theorem 4 of part I, we know that all potential h-arcs are still compatible in a general convex polygon. However, unlike those in a monotone polygon, the potential h-arcs no longer form a linear list. Instead, they form a tree, called an arc-tree. In Fig. 11a, there is a 12-gon with 6 potential h-arcs and they are labelled as h_2, h_3, h_4, h_5, h_6, and h_7. (Note that we also obtain V_4-V_3, V_7-V_6 and V_6-V_8 from the one-sweep algorithm. In order to have a simpler example, let us assume that all these three arcs are unstable and hence are not shown in Fig. 11a.) To get a better feeling of the arc-tree, we can redraw the 12-gon as shown in Fig. 11b. Again, we regard V_1 as a degenerated arc h_1, V_{12} as a degenerated arc h_8, and V_{11} as a degenerated arc h_9.

The father-son relationship still holds for the h-arcs in a general polygon, and we can also define supporting weights of the arcs in a similar way. The only difference is that the ceiling of a subpolygon may consist of more than one arc. Before we can calculate the supporting weight of any arc, we must process all the arcs above it, i.e. all the arcs in its upper subpolygon. Hence, we can do a post-order traversal through the arc tree. Let us consider the following two examples. Again, for simplicity, we assume that all arcs have distinct positions in the examples.



(b)

Fig. 11. A general 12-gon.

Example 3.

We first compare

$$H_0\begin{pmatrix}h_8\\h_5\end{pmatrix} + H_0\begin{pmatrix}h_5\\f_5\end{pmatrix} = H_0\begin{pmatrix}h_8\\f_5\end{pmatrix}$$

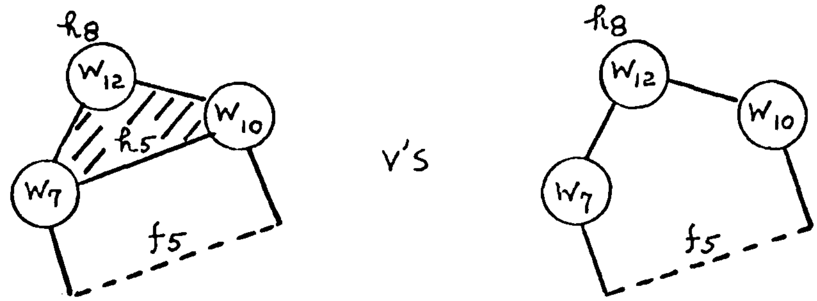


Fig. 12. Illustrations for Example 3.
12a. To find f_5.

Assume h_4/f_5, we compare

$$H_0\begin{pmatrix}h_5\\h_4\end{pmatrix} + H_0\begin{pmatrix}h_4\\f_4\end{pmatrix} = H_0\begin{pmatrix}h_5\\f_4\end{pmatrix}$$

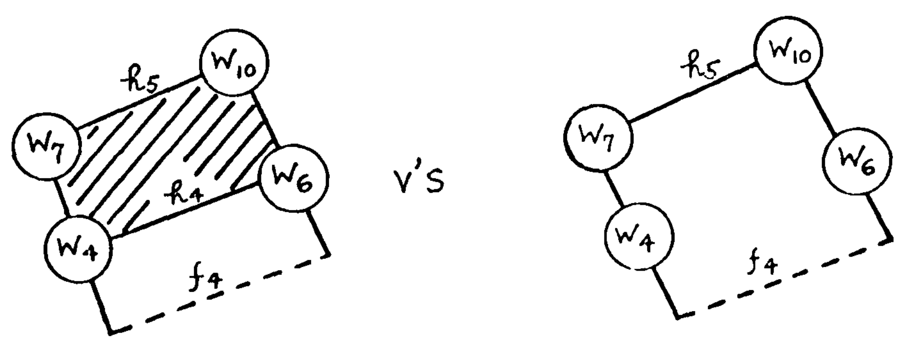


Fig. 12b. To find f_4.

Assume h_3/f_4 and f_5/f_4, we condense h_5 into h_4,

$$H_0\binom{h_8}{h_5} + H_0\binom{h_5}{h_4} + H_0\binom{h_4}{f_{54}} = H_0\binom{h_8}{f_{54}}$$

or

$$C\binom{h_8}{h_4} + H_0\binom{h_4}{f_{54}} = H_0\binom{h_8}{f_{54}}$$

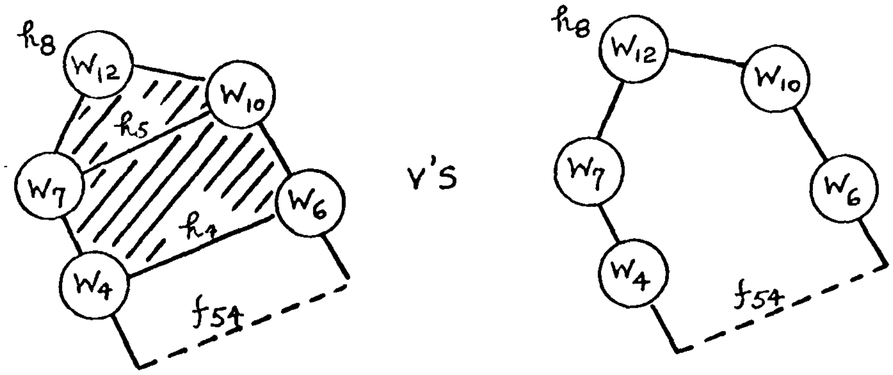


Fig. 12c. To find f_{54}.

Before we can process h_3, we have to process h_7 and h_6 first. Hence, the next comparison is:

$$H_0\binom{h_9}{h_7} + H_0\binom{h_7}{f_7} = H_0\binom{h_9}{f_7}$$

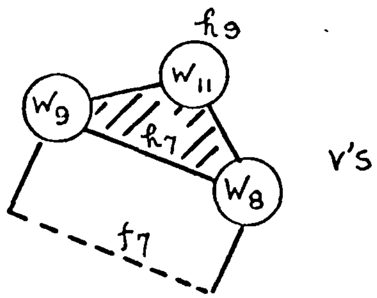


Fig. 12d. To find f_7.

Assume h_6/f_7, we compare

$$H_0\begin{pmatrix} h_7 \\ h_6 \end{pmatrix} + H_0\begin{pmatrix} h_6 \\ f_6 \end{pmatrix} = H_0\begin{pmatrix} h_7 \\ f_6 \end{pmatrix}$$

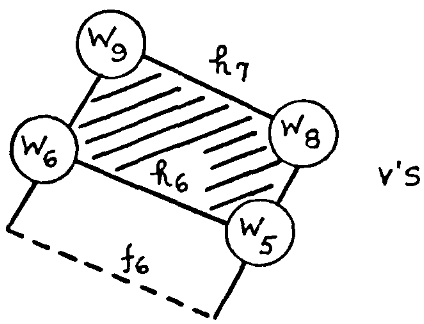


Fig. 12e. To find f_6.

We have h_3/f_6 and f_7/f_6, we condense h_7 into h_6,

$$H_0\begin{pmatrix} h_9 \\ h_7 \end{pmatrix} + H_0\begin{pmatrix} h_7 \\ h_6 \end{pmatrix} + H_0\begin{pmatrix} h_6 \\ f_{76} \end{pmatrix} = H_0\begin{pmatrix} h_9 \\ f_{76} \end{pmatrix}$$

or

$$C\begin{pmatrix} h_9 \\ h_6 \end{pmatrix} + H_0\begin{pmatrix} h_6 \\ f_{76} \end{pmatrix} = H_0\begin{pmatrix} h_9 \\ f_{76} \end{pmatrix}$$

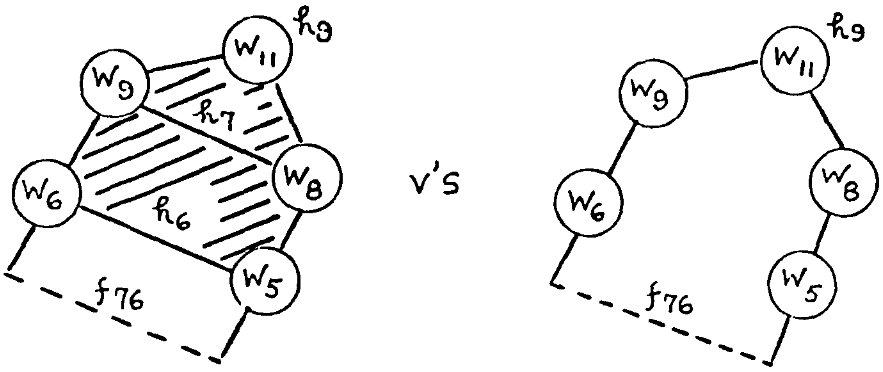


Fig. 12f. To find f_{76}.

Assume h_3/f_{76} and next we process the arc h_3, using both h_4 and h_6 as the ceilings of h_3,

$$H_0\begin{pmatrix} h_4, h_6 \\ h_3 \end{pmatrix} + H_0\begin{pmatrix} h_3 \\ f_3 \end{pmatrix} = H_0\begin{pmatrix} h_4, h_6 \\ f_3 \end{pmatrix}$$

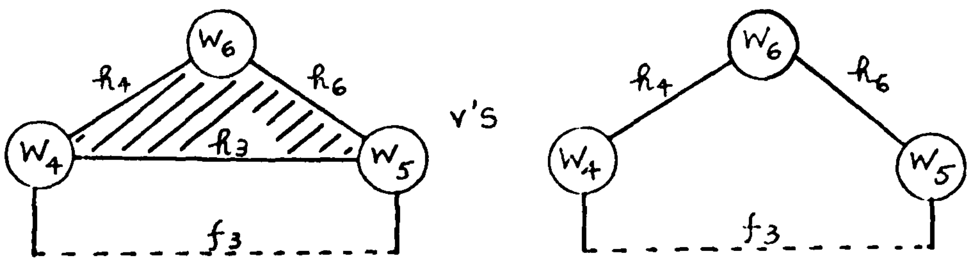


Fig. 12g. To find f_3.

Suppose h_2/f_3 and $f_{54}/f_{76}/f_3$, we first condense h_5 and h_4 into h_3 and we get

$$C\begin{pmatrix} h_8, h_6 \\ h_3 \end{pmatrix} + H_0\begin{pmatrix} h_3 \\ f_{544} \end{pmatrix} = H_0\begin{pmatrix} h_8, h_6 \\ f_{543} \end{pmatrix}$$

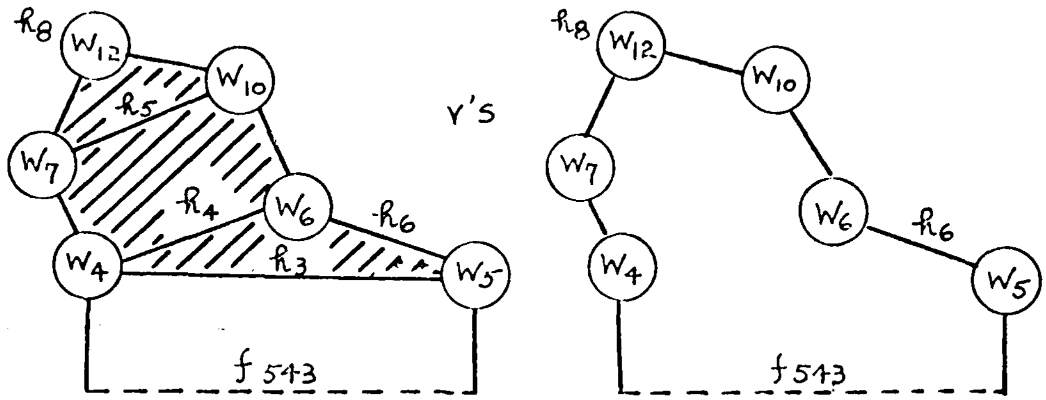


Fig. 12h. To find f_{543}.

Now, h_2/f_{543} and f_{76}/f_{543}, so we condense h_7 and h_6 into h_3 and obtain

$$C\begin{pmatrix} h_8, h_9 \\ h_3 \end{pmatrix} + H_0\begin{pmatrix} h_3 \\ f_{54763} \end{pmatrix} = H_0\begin{pmatrix} h_8, h_9 \\ f_{54763} \end{pmatrix}$$

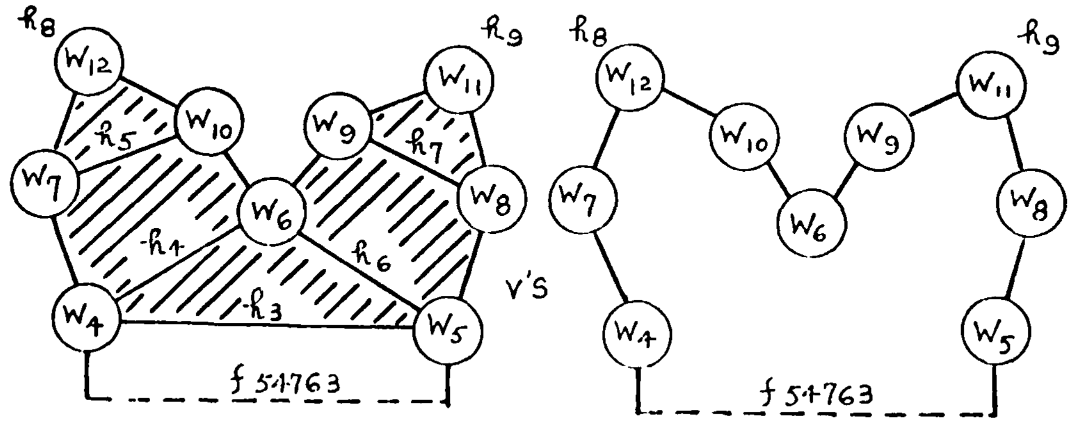


Fig. 12i. To find f_{54763}.

Assume h_2/f_{54763} and we compare

$$H_0\begin{pmatrix} h_3 \\ h_2 \end{pmatrix} + H_0\begin{pmatrix} h_2 \\ f_2 \end{pmatrix} = H_0\begin{pmatrix} h_3 \\ f_2 \end{pmatrix}$$

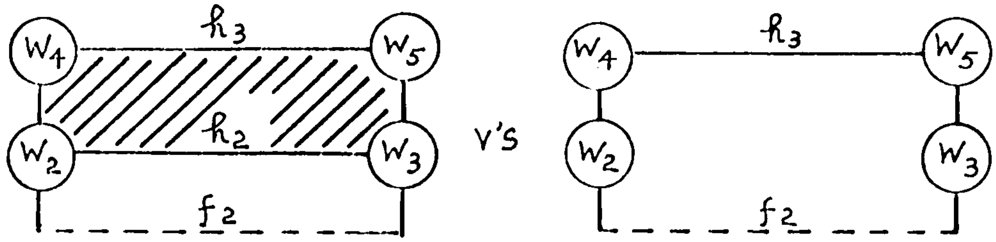


Fig. 12j. To find f_2.

Suppose h_1/f_2 and f_{54763}/f_2, we condense h_3 and its descendants into h_2 and get

$$C\begin{pmatrix} h_8, h_9 \\ h_2 \end{pmatrix} + H_0\begin{pmatrix} h_2 \\ f_{547632} \end{pmatrix} = H_0\begin{pmatrix} h_8, h_9 \\ f_{547632} \end{pmatrix}$$

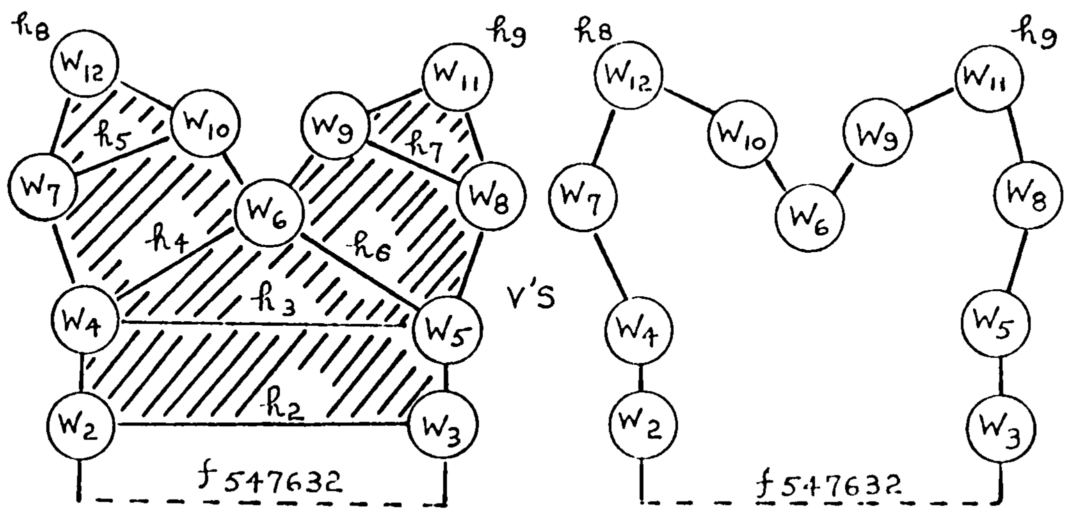


Fig. 12k. To find f_{547632}.

If h_1/f_{547632}, the ℓ-optimum partition of the whole polygon consists of all six h-arcs h_2, h_3, h_4, h_5, h_6, and h_7. If f_{547632}/h_1, all six h-arcs will be removed and the ℓ-optimum partition is a fan.

Example 4

We first compare

$$H_0\binom{h_8}{h_5} + H_0\binom{h_5}{f_5} = H_0\binom{h_8}{f_5}$$

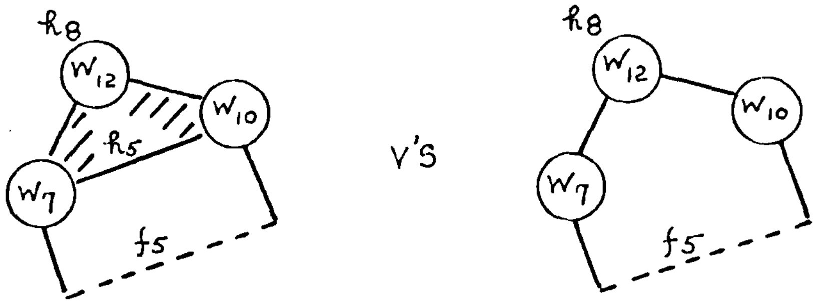


Fig. 13. Illustrations for Example 4.
13a. To find f_5.

Assume h_4/f_5 and we compare

$$H_0\binom{h_5}{h_4} + H_0\binom{h_4}{f_4} = H_0\binom{h_5}{f_4}$$

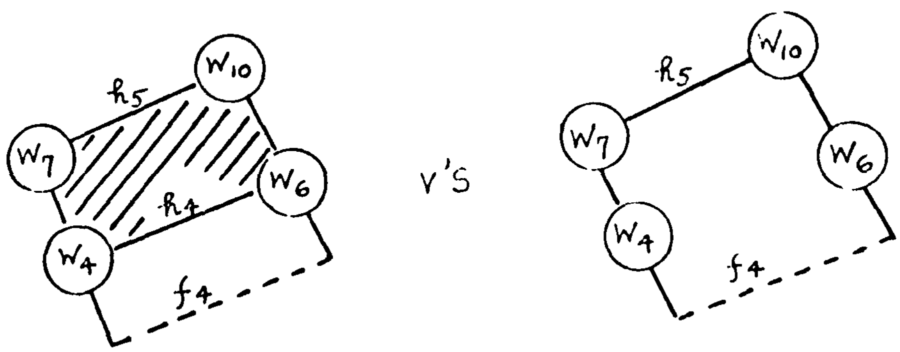


Fig. 13b. To find f_4.

Let h_3/f_4 and f_4/f_5, so we compare

$$H_0\binom{h_9}{h_7} + H_0\binom{h_7}{f_7} = H_0\binom{h_9}{f_7}$$

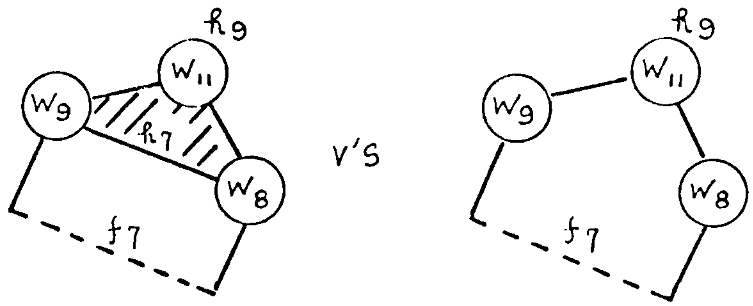


Fig. 13c. To find f_7.

h_6/h_7 and we compare

$$H_0\binom{h_7}{h_6} + H_0\binom{h_6}{f_6} = H_0\binom{h_7}{f_6}$$

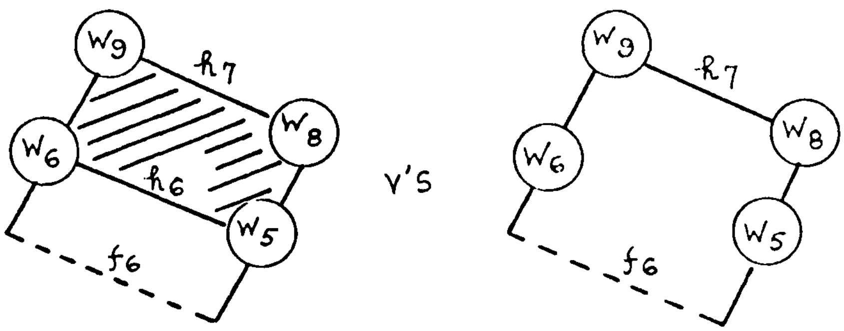


Fig. 13d. To find f_6.

We have h_3/f_6 and f_6/f_7, so our next comparison will be

$$H_0\begin{pmatrix} h_4, h_6 \\ h_3 \end{pmatrix} + H_0\begin{pmatrix} h_3 \\ f_3 \end{pmatrix} = H_0\begin{pmatrix} h_4, h_6 \\ f_3 \end{pmatrix}$$

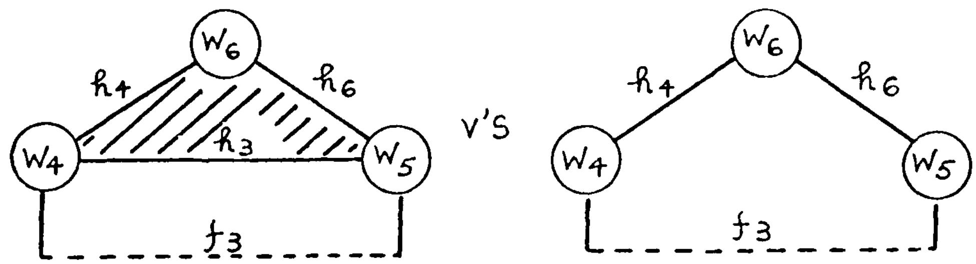


Fig. 13e. To find f_3.

Suppose h_2/f_3 and $f_4/f_3/f_6$, we condense h_4 into h_3

$$C\begin{pmatrix} h_5, h_6 \\ h_3 \end{pmatrix} + H_0\begin{pmatrix} h_3 \\ f_{43} \end{pmatrix} = H_0\begin{pmatrix} h_5, h_6 \\ f_{43} \end{pmatrix}$$

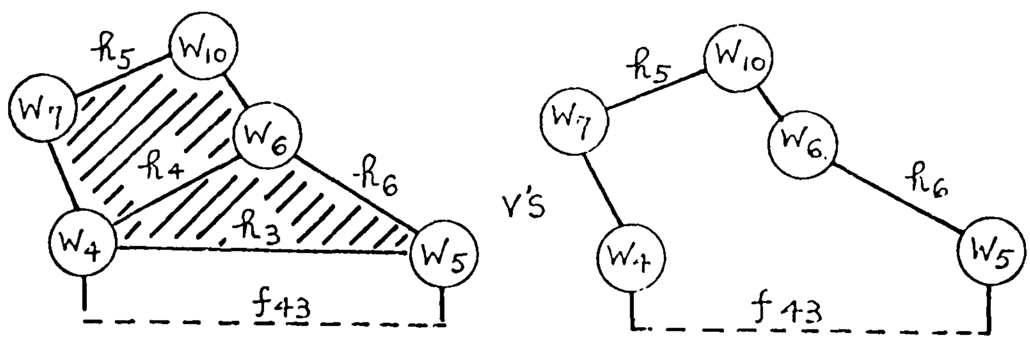


Fig. 13f. To find f_{43}.

Assume h_2/f_{43}, f_{43}/f_6, and f_{43}/f_5, we proceed to process h_2,

$$H_0\begin{pmatrix} h_3 \\ h_2 \end{pmatrix} + H_0\begin{pmatrix} h_2 \\ f_2 \end{pmatrix} = H_0\begin{pmatrix} h_3 \\ f_2 \end{pmatrix}$$

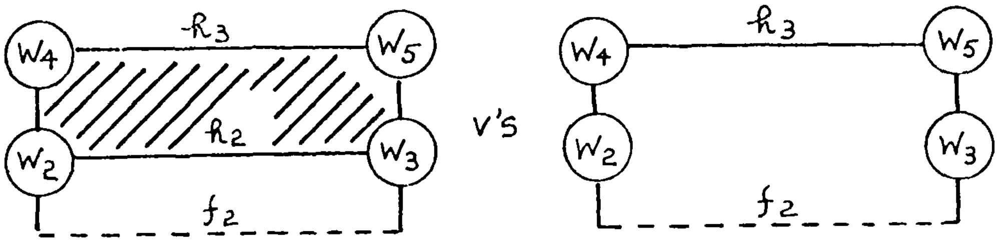


Fig. 13g. To find f_2.

Assume h_1/f_2 and f_{43}/f_2, we condense h_3 into h_2,

$$C\begin{pmatrix} h_5, h_6 \\ h_2 \end{pmatrix} + H_0\begin{pmatrix} h_2 \\ f_{432} \end{pmatrix} = H_0\begin{pmatrix} h_5, h_6 \\ f_{432} \end{pmatrix}$$

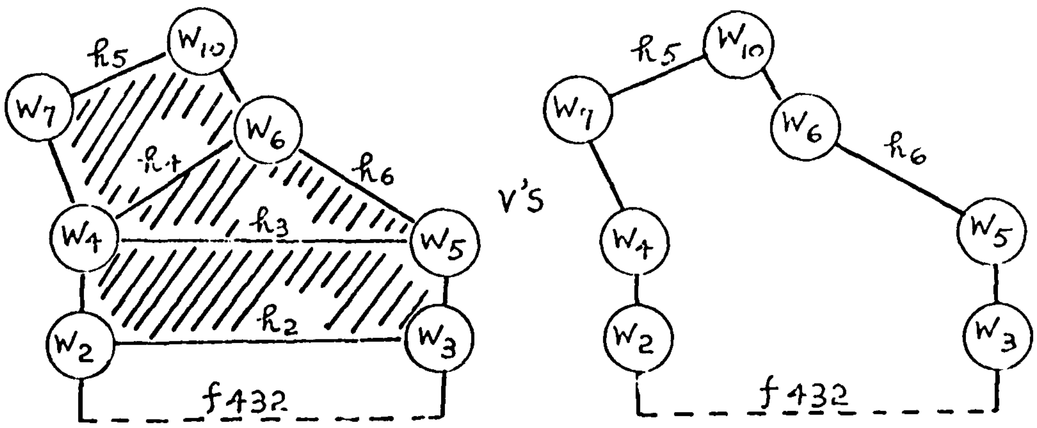


Fig. 13h. To find f_{432}.

Suppose f_{432}/h_1, we remove h_2 as well as its descendants h_3 and h_4. Assume f_6/f_5 and f_6/h_1, we remove h_6 from the polygon. Now, we have f_7/f_5 and f_7/h_1, so we remove h_7 from the polygon. Finally, we have h_1/f_5, and the ℓ-optimum partition of the polygon consists of one h-arc h_5.

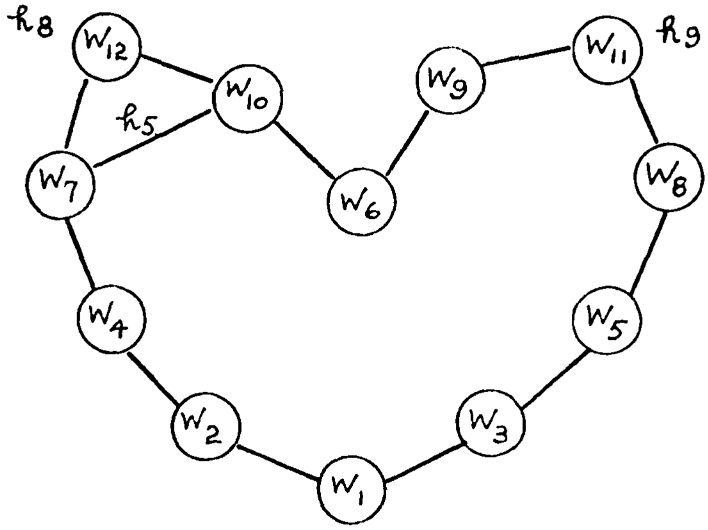


Fig. 13i. The optimum partition.

From the above two examples, we have the following observations.

(1) Before we can process a potential h-arc, we have to process all the arcs above it. Hence, we should do a post-order traversal, starting at the root of the arc tree, i.e. the degenerated arc h_1.

(2) Whenever we do a condensation or deletion, we always pick the ceiling arc which has the highest floor first, i.e. the one with the largest supporting weight. Hence, we should keep track of the order of the ceiling arcs.

(3) Once a ceiling arc h_j of h_i is removed or condensed, the ceiling arcs of h_j become the ceiling arcs of h_i and we have to update the order of all the ceiling arcs of h_i.

One way of keeping track of the order of the ceiling arcs is to keep them in a priority queue.

Now, let us outline the algorithm for finding the optimum partition of a general convex polygon.

1. Get all the potential h-arcs of the polygon by the one-sweep algorithm. (All the h-arcs form a tree, with the root at the bottom. Let the arc-tree be T.)

2. Process the h-arcs, one by one, from the leaves to the root. (We always process the children before we process the father, and we always obtain the optimum partition of the subpolygon bounded below by the arc being processed.)

3. Let h_R be the arc currently being examined, U_R be the set of arcs immediately above h_R, and h_N be the arc immediately below h_R in T. If h_R is negative in the subpolygon bounded above by the arcs in U_R and below by h_N, delete h_R, else go to step 5.

4. Once h_R and its descendants are deleted, we examine the arcs in U_R to see if we can delete more arcs. If yes, we delete the arc with the largest supporting weight; then we include its celing arcs into U_R and repeat this step. Otherwise, we move to process the next arc.

5. Now, h_R is positive in the smallest subpolygon. If there exists some arc in U_R, say h_j, such that

$$S\binom{U_R}{h_R} < \text{the supporting weight of } h_j ,$$

we will pick the arc with the largest supporting weight in U_R, condense it with its descendants into h_R and include all its ceiling arcs into U_R. Then we compare the cost of the partition with h_R and its descendants against that of the fan in the subpolygon bounded above by the arcs in U_R and below by h_N. If the fan is ℓ-optimum in the subpolygon, we remove h_R as well as all its descendants from T, and we examine the arcs in U_R to see if we can delete any more arcs. Otherwise, we examine the arcs in U_R to see if we can condense any more arcs.

6. Now, $S\binom{U_R}{h_R} \geq$ the supporting weight of every arc in U_R. The arcs in U_R are the ceiling arcs of h_R and $S\binom{U_R}{h_R}$ is the supporting weight of h_R. We move to process the next arc.

Before presenting the algorithm, let us describe a recursive procedure to process the potential h-arcs of any subpolygon.

Procedure PARTITION (ROOT)

Input: consists of a set of potential h-arcs of a subpolygon. These arcs are arranged in the form of an arc tree, like the one shown in Fig. 11b. The root of the tree is passed to the procedure via the argument ROOT.

Output: consists of a set of the potential h-arcs which appear in the ℓ-optimum partition of the subpolygon. We can divide that arcs into two types: (i) those arcs which are descendants of some other arcs in the set and (ii) those arcs which have no ancestor in the set. The arcs in type (i) are condensed into their ancestors and can be traced out from the arcs in type (ii). The arcs in type (ii) are called ceiling arcs and are kept in a reduced arc tree. The root of the arc tree is passed back to the caller via the parameter ROOT.

Step 0 Let the arc at the root of the input arc tree be h_N ;

MIN-WEIGHT := the weight of the minimum of the two vertices of h_N ;

T := an arc tree with only one arc, h_N ;

Step 1 For each arc immediately above h_N in the input arc-tree Do

Begin

Step 1a Let the arc to be processed be h_R ;

If there exists a non-degenerated arc above h_R

then go to Step 1b

else go to Step 1f;

Comment: h_R is immediately above h_N .

Step 1b PARTITION (h_R) ;

Let the subtree returned be T' ;

Comment: Before processing h_R , the subtrees of h_R are first processed recursively.

Step 1c Let U_R be the set of arcs immediately above h_R in T' ;

Calculate $S\begin{pmatrix} U_R \\ h_R \end{pmatrix}$;

If $S\begin{pmatrix} U_R \\ h_R \end{pmatrix} \geq$ MIN-WEIGHT

then go to Step 1d

else go to Step 1e.

Step 1d   Remove h_R from T';

while (there exists a non-degenerated arc, h_j, in U_R) and (the supporting weight of $h_j \geq$ MIN-WEIGHT) Do

Begin

Remove h_j from U_R;

Remove h_j from T';

Include all ceiling arcs of h_j into U_R;

end;

Insert the forest T' into T such that all arcs in U_R are immediately above h_N in T.

Go to Step 1i.

Step 1e   If (there exists a non-degenerated arc in U_R) and (its supporting weight > the supporting weight of h_R)

then

Begin

Among all the arcs in U_R, pick the one with maximum supporting weight;

Let it be h_j;

Condense h_j into h_R and remove it from T';

Include all ceiling arcs of h_j into U_R;

Fix up the tree T' so that all the ceiling arcs of h_j are immediately above h_R in T';

go to Step 1c;

end

else

Begin

Record $S\binom{U_R}{h_R}$ as the supporting weight of h_R and all arcs in U_R as the ceiling arcs of h_R; insert T' into T so that h_R is immediately above h_N in T;

go to Step 1i ;

end.

Step 1f Let h_C be the degenerated arc above h_R ;

Calculate $S\binom{h_C}{h_R}$;

If $S\binom{h_C}{h_R} \geq$ MIN-WEIGHT

then go to Step 1g

else go to Step 1h .

Step 1g Remove h_R ;

Insert h_C immediately above h_N in T.

go to Step 1i .

Step 1h Record $S\binom{h_C}{h_R}$ as the supporting weight of h_R and h_C as the ceiling arc of h_R; insert the subtree with h_R and h_C into T so that h_R is immediately above h_N in T.

Step 1i End.

Step 2. Return T with root stored in ROOT to caller.

Now, the details of the algorithm to find an optimum partition of a convex polygon is presented.

Algorithm II

Input consists of n positive integers, which are the weights of the n vertices of an n-gon. $W[1]$ is the weight of the minimum vertex and $W[i+1]$ is the neighbor of $W[i]$ of the n-gon going in the clockwise direction.

Output consists of a tree of potential h-arcs which exist in the ℓ-optimum partition of the n-gaon.

Step 0 For $i := 2$ step 1 until N do

$$CP[i] := \sum_{j=1}^{i-1} W[j] \cdot W[j+1] ;$$

$CP[1] := 0$;

Comment: The sum of adjacent products $W[i] : W[j]$ can be obtained from $CP[j] - CP[i]$ for $1 \leq i < j < N$.

Step 1 Apply the one-sweep algorithm to obtain a tree of arcs. Let this arc tree be T.

Comment: T contains all potential h-arcs in the n-gon.

Step 2 From T, remove those arcs which are not potential h-arcs;

If T is empty

then go to Step 6.

else go to Step 3.

Step 3 Insert the degenerated arc h_1 with weight $W[1]$ to the bottom of the tree, as the root of the tree;

Insert a degenerated arc with the local maximum weight at the tip of each corresponding branch of the arc tree.

Step 4 PARTITION (h_1);

Comment: h_1 is the root of T; when returned from PARTITION, T will contain all the ceiling arcs with their descendants in the ℓ-optimum partition.

Step 5 Remove all degenerated arcs.

Step 6 Output T and stop.

This algorithm has been implemented in Pascal and the listing of the computer program is given in Appendix II.

Theorem 3. The partition of the general convex n-gon obtained by the algorithm is ℓ-optimum.

Proof. Using arguments similar to those in Theorem 2, we can first prove that all the potential h-arcs which are deleted from the arc-tree cannot be present in the ℓ-optimum partition, then we prove that any arc which is left in the arc-tree at the end of the algorithm must be present in the ℓ-optimum partition. Hence, the partition consisting of the h-arcs output by the algorithm and with fans in the subpolygons bounded by a potential h-arc and the arcs immediately above it in the output arc-tree must be ℓ-optimum. ■

Let us examine how much time we spend in executing the algorithm.

Steps 0 and 1 each scans the polygon once, and hence takes $O(n)$ time. Since there are at most $n-3$ arcs in T, Step 2 also takes $O(n)$ time. There will be a recursive procedure call for each arc in T (except the leaf nodes). Inside each procedure call, there are two nested loops. The innermost loop is the "while" loop in Step 1d and the outer one spans from Steps 1c to 1e. Whenever the innermost loop is executed once, a potential h-arc is deleted from T. Whenever the outer loop is executed once (i.e. the "then" part of Step 1e), a potential h-arc is condensed into its father. Once an arc is deleted or condensed, it will never be examined again. In order to carry out the deletion and condensation efficiently, we cannot examine all the arcs in U_R each time we go through the loop. Hence, we need to order the arcs in U_R in a priority queue and it takes $O(\log n)$ to update the queue each time. Hence, it takes $O(n \log n)$ time in executing Step 4 of the algorithm. Steps 5 and 6 each takes $O(n)$ time. Hence, the whole algorithm takes $O(n \log n)$ time to find the ℓ-optimum partition.

4. A closer look at the optimum partitions

We now present some theorems which enable the algorithm to divide the polygon into several subpolygons and hence can improve the average performance of the algorithm. These theorems have also been mentioned in [4] without detailed proofs.

Let us consider the polygons where there are two or more vertices with equal weights w_1.

Lemma 9. For every choice of $V_1, V_2, \ldots$ (as prescribed in Part I), if the weights of the vertices satisfy the condition

$$w_1 = w_2 < w_3 \leq \cdots \leq w_n \quad ,$$

then V_1-V_2 exists in every optimum partition of the n-gon.

Proof. The lemma is true if V_1-V_2 is a side of the n-gon. Hence, we can assume that V_1, V_2 are not adjacent to the same side of the n-gon.

The proof is by induction on the size of the n-gon. The lemma is true for a triangle and a quadrilateral. Assume that the lemma is true for all k-gons $(3 \leq k \leq n-1)$ and consider the optimum partitions of an n-gon.

By Lemma 3 of Part I, we know that there are at least two vertices with degree two in each optimum partition of the n-gon. We have the following two cases.

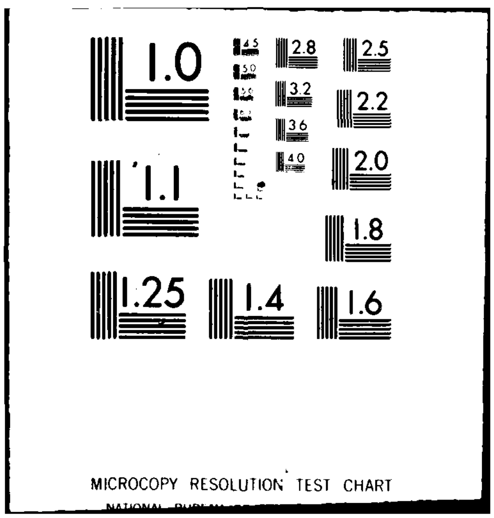
(i) In an optimum partition of an n-gon, one of the vertices with degree two, say V_i, has weights larger than w_1. In this case, we can form an (n-1)-gon by removing V_i with its two sides. By induction assumption, V_1-V_2 is present in every optimum partition of the (n-1)-gon.

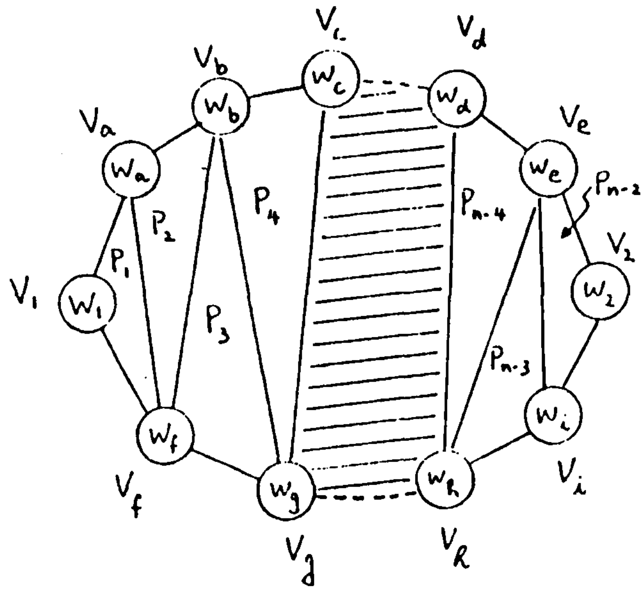
(ii) Consider the complementary case of (i), i.e. all vertices with degree two have weights equal to w_1 in an optimum partition of the n-gon. In other words, V_1 and V_2 are the only two vertices with degree two in that optimum partition, as shown symbolically in Fig. 14a. Note that every arc in the optimum partition must dissect the n-gon into two subpolygons in such a way that V_1, V_2 can never appear in any subpolygon together, else there will be more than two vertices with degree two in the optimum partition. In Fig. 14b we show a partition of the n-gon in which V_1 and V_2 are connected. Let us denote the n-2 triangles in Fig. 14a by $P_1, P_2, \ldots, P_{n-2}$. Except P_1 and P_{n-2}, all the other n-4 triangles are made up of one side and two arcs each. For each of these n-4 triangles, we can find a unique triangle in Fig. 14b such that they both consist of the same side. We use P_i' to denote the image of P_i in Fig. 14b. The only two triangles left unmatched in Fig. 14b are $V_1V_aV_2$ and $V_1V_2V_i$ and they are the images of P_1 and P_{n-2}, respectively. Let the cost of P_i be C_i and the cost of P_i' be C_i'. Since $C_i' < C_i$ for $1 \leq i \leq n-2$, the partition in Fig. 14b is cheaper than that in Fig. 14a and we have contradiction. ■

AD-A113 349 STANFORD UNIV CA DEPT OF COMPUTER SCIENCE F/G 12/1
COMPUTATION OF MATRIX CHAIN PRODUCTS. PART I, PART II.(U)
SEP 81 T C HU, M T SHING DAAG29-80-C-0029
UNCLASSIFIED STAN-CS-81-875 ARO-16323.2-M NL

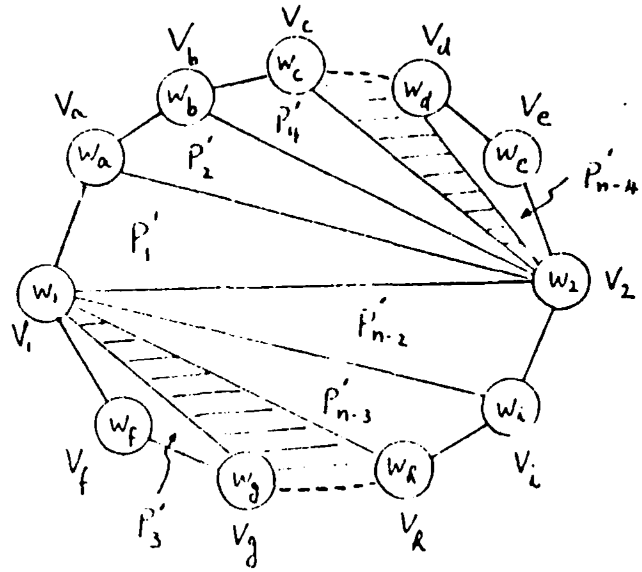
2 of 2

END
DATE
FILMED
5-82
DTIC


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART



(a)



(b)

Fig. 14

Theorem 4. For every choice of $V_1, V_2, \ldots$ (as prescribed in Part I), if the weights of the vertices satisfy the condition

$$w_1 = w_2 < w_3 \leq w_4 \leq \cdots \leq w_n \ ,$$

then every optimum partition of the n-gon must contain a triangle $V_1V_2V_p$ for some vertex V_p with weight equal to w_3. Note that if $w_1 = w_2 < w_3 < w_4 \leq \ldots \leq w_n$, then every optimum partition must contain the triangle $V_1V_2V_3$ since there is a unique choice of V_3.

Proof. Similar to Lemma 9, we can prove this theorem by induction on the size of the n-gon. The theorem is true for any triangle or quadrilateral satisfying the above condition. Assume the theorem is true for all k-gons $(3 \leq k \leq n-1)$ and consider the optimum partitions of an n-gon.

From Lemma 9, we know that V_1, V_2 are always connected in every optimum partition. Hence, without loss of generality, we can assume V_1, V_2 to be adjacent to the same side of the n-gon. Again, we have the following two cases.

(i) In an optimum partition, one of the vertices with degree two, say V_i, has weight larger than w_3. In this case, we can remove V_i with its sides and form an (n-1)-gon. By induction assumption, every optimum partition of the (n-1)-gon contains a triangle $V_1V_2V_p$ for some vertex V_p where $w_p = w_3$.

(ii) Consider the complementary case of (i), in an optimum partition of the n-gon, all vertices with degree two have weights less than or equal to w_3. Since V_1-V_2 is a side of the n-gon, for $n \geq 4$, either V_1 or V_2 (but not both) can have degree two. We have the following two subcases:

(a) If there are more than one vertex whose weight equals w_3, we can form an (n-1)-gon by removing one of those degree two vertices whose weight equals w_3. By induction assumption, every optimum partition of the (n-1)-gon contains a triangle $V_1V_2V_p$ for some vertex V_p with $w_p = w_3$,

(b) There exists only one vertex of weight w_3. In this case, there must be only two vertices with degree two in the optimum partition of the n-gon. These two vertices are V_3 and either V_1 or V_2. Without loss of generality, we can assume V_1 has degree 2. The situation is shown symbolically in Fig. 15a. Again, every arc in the optimum partition must dissect the n-gon in such a way that V_1 and V_3 can never appear in any subpolygon together. In Fig. 15b, we show a partition containing the triangle $V_1V_2V_3$. Using arguments similar to those in the proof of Lemma 9, we can show that the partition in Fig. 15b is cheaper and we obtain a contradiction. ∎

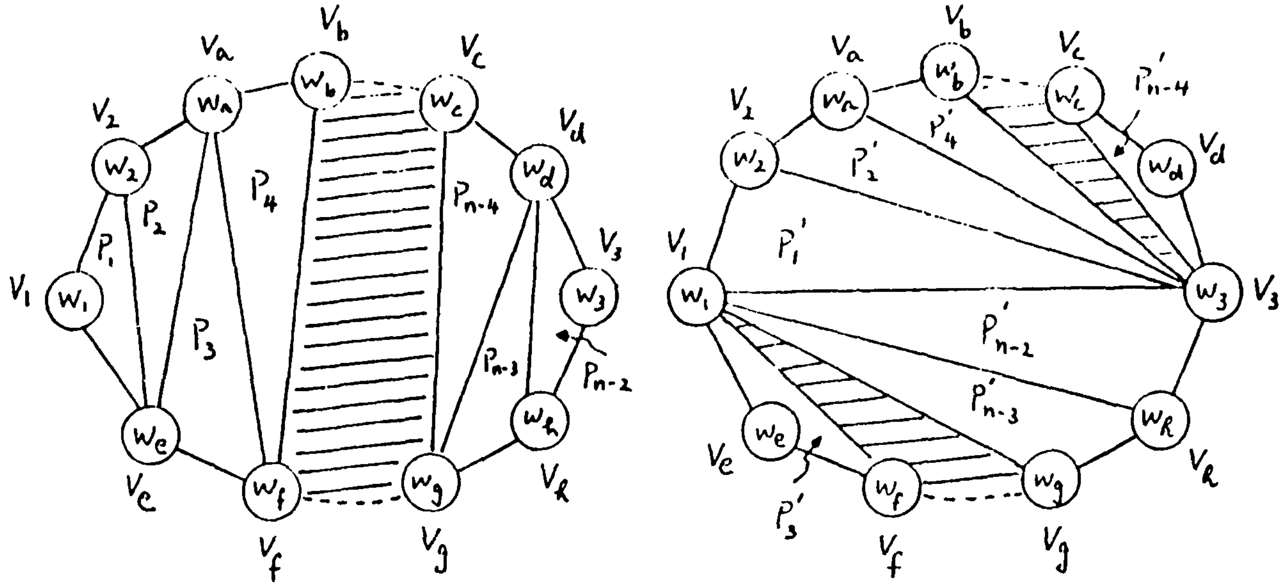


Fig. 15

<u>Theorem 5</u>. For every choice of $V_1, V_2, \ldots$ (as prescribed in Part I), if the weights of the vertices of the n-gon satisfy the following condition,

$$w_1 = w_2 = \cdots = w_k < w_{k+1} \leq \cdots \leq w_n$$

for some k, $3 \leq k \leq n$, then every optimum partition of the n-gon contains the k-gon $V_1 - V_2 - \cdots - V_k$.

<u>Proof.</u> The proof is by induction on th[illegible] of the n-gon. The theorem is true for any triangle and quadrilateral. Suppose the theorem is true for all polygons with (n-1) sides or less and consider the optimum partitions of an n-gon.

From Lemma 3 of Part I, there exist at least two vertices having degree two in every optimum partition. We have the following two cases.

(i) In an optimum partition of the n-gon, one of the vertices with degree two, say V_i, has weight larger than w_1. In this case, we can remove the vertex V_i with its two sides and obtain an (n-1)-gon. By induction assumption, every optimum partition of the (n-1)-gon contains the k-gon $V_1-V_2-\dots-V_k$.

(ii) Consider the complementary case of (i), i.e., all the vertices with degree two have weights equal to w_1 in an optimum partition. Let two of these vertices be V_i, V_j. We have the following two subcases:

(a) $k > 3$. We first form an (n-1)-gon by removing V_i and its two sides. There are (k-1) vertices with weights equal to w_1 in the (n-1)-gon. By induction assumption, every optimum partition of the (n-1)-gon contains the (k-1)-gon which includes V_j as one of its vertices. Since V_j has degree two in the optimum partition, its two neighboring vertices, say V_x and V_y, must also have weights equal to w_1 and the arc V_x-V_y exists in the optimum partition (Fig. 16). Similarly, we can remove the vertex V_j with its two sides V_j-V_x and V_j-V_y and form an (n-1)-gon. By induction assumption, every optimum partition of the (n-1)-gon contains the (k-1)-gon formed by the (k-1) vertices with weights equal to w_1 in the (n-1)-gon and V_i is one of the vertices in the (k-1)-gon. Now, by pasting the triangle $V_xV_jV_y$ and the (k-1)-gon together, we form a k-gon which includes all the vertices with weight equal to w_1 in the n-gon and this k-gon is contained in the optimum partition of the n-gon.

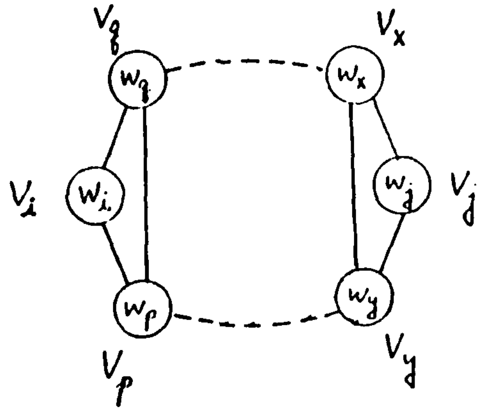


Fig. 16

(b) $k = 3$. In this case, we have $w_1 = w_2 = w_3 < w_4 \le \cdots \le w_n$. Without loss of generality, we can assume V_1 and V_2 both have degree two in an optimum partition. Again, we can form an (n-1)-gon by removing V_1 and its two sides. By Lemma 9, V_2 and V_3 are connected in every optimum partition of the (n-1)-gon. Since V_2 has degree two, V_2-V_3 must be a side of the n-gon. Next, we can remove V_2 with its two sides and form an (n-1)-gon. By Lemma 9, V_1, V_3 are connected by a side of the n-gon. The situation is shown in Fig. 17a. Then, the partition in Fig. 17b is cheaper because

$$T_{123} + T_{12y} \le T_{13x} + T_{23y} \ .$$

and $$C(w_1, w_x, \ldots, w_y) < C(w_3, w_x, \ldots, w_y) \ . \quad \blacksquare$$

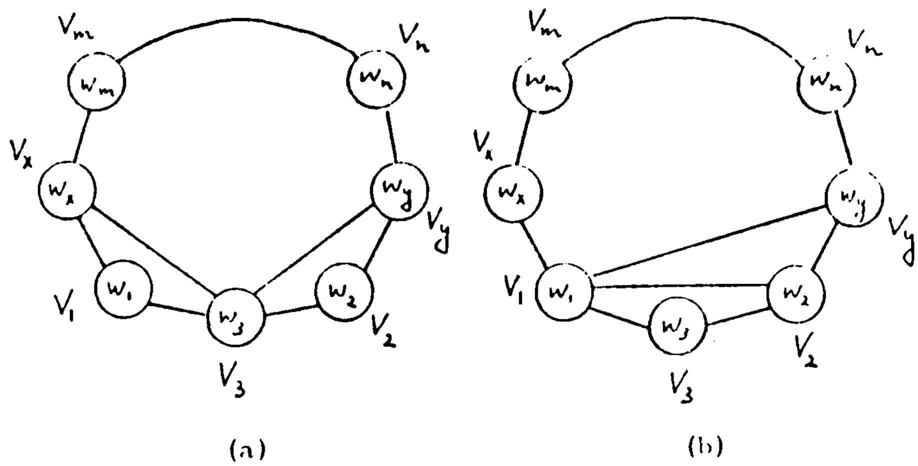


Fig. 17

Now, whenever we have three or more vertices with weights equal to w_1 in the n-gon, we can decompose the n-gon into subpolygons by forming the k-gon in Theorem 5. The partition of the k-gon can be arbitrary, since all vertices of the k-gon are of equal weight. For any subpolygon with two vertices of weights equal to w_1, we can always apply Theorem 4 and decompose the subpolygon into smaller subpolygons. Hence, we have only to consider the polygons with a unique choice of V_1, i.e., each polygon has only one vertex with weight equal to w_1.

Because of Theorems 4 and 5, Theorems 1 and 3 of Part I can be generalized as follows.

Theorem 6. For every choice of $V_1, V_2, \ldots$ (as prescribed in Part I), if the weights of the vertices satisfy the condition

$$w_1 < w_2 \le w_3 \le \cdots \le w_n ,$$

then V_1-V_2 and V_1-V_3 exist in every optimum partition of the n-gon. ■

Theorem 7. Let V_x and V_z be two arbitrary vertices which are not adjacent in a polygon, and V_w be the smallest vertex from V_x to V_z in the clockwise manner ($V_w \neq V_x$, $V_w \neq V_z$), and V_y be the smallest vertex from V_z to V_x in the clockwise manner ($V_y \neq V_x$, $V_y \neq V_z$). This is shown in Fig. 18 where we assume that $V_x < V_z$ and $V_y < V_w$. The necessary condition for V_x-V_z to exist as an h-arc in any optimum partition is

$$w_y < w_x \leq w_z < w_w \ . \qquad \blacksquare$$

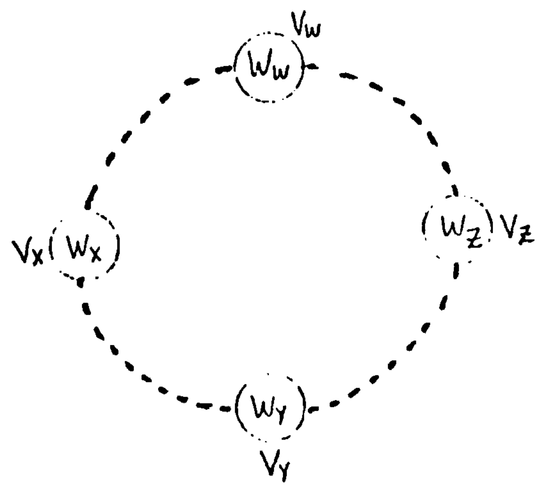


Fig. 18

From Theorem 7, we know that any arc which exists as an h-arc in some optimum partition must be a potential h-arc. In other words, the h-arcs in every optimum partition will be generated by the one-sweep algorithm. Hence, by modifying the condition in steps 1c and 1d of the procedure Partition to favor partitions with more h-arcs, we can obtain other optimum partitions which consist of more h-arcs than the ℓ-optimum partition.

5. Conclusion

The problem to find the optimum order of computing a chain of matrices has been around for several years [2]. It has been used as a typical example to illustrate the dynamic programming technique in many textbooks [1][3]. In this paper, a new approach is used to solve the problem. Instead of tackling the matrix chain product problem directly, it is transformed into the problem of partitioning a convex polygon and a tailor-made algorithm for finding the optimum partition is developed. The algorithm takes O(n log n) time and O(n) space. For those who want to trade optimum solution for shorter execution time, an O(n) heuristic algorithm has been presented in [5]. This heuristic algorithm is very simple to implement and its error bound given explicitly as a function of the number of sides of the convex polygon and the ratio of the weights of the largest vertex to that of the smallest vertex. The worst error ratio is less than 15%.

References

1. A. V. Aho, J. E. Hopcroft and J. D. Ullman, "The Design and Analysis of Computer Algorithms," Addison-Wesley, 1974.

2. S. S. Godbole, "An Efficient Computation of Matrix Chain Products," IEEE Trans. Computers C-22, 9 Sept. 1973, pp. 864-866.

3. H. Horowitz and S. Sahni, "Fundamentals of Computer Algorithms," Computer Science Press, 1978, pp. 242-243.

4. T. C. Hu and M. T. Shing, "Computation of Matrix Chain Products," 1981 Army Numerical Analysis and ComputerConferences, February 1981.

5. T. C. Hu and M. T. Shing, "An O(n) Algorithm to Find a Near-Optimum Partition of a Convex Polygon," to appear in the Journal of Algorithms.

Appendix I

```
PROGRAM OPTIMUM_ALGORITHM_FOR_A_MONOTONE_BASIC_POLYGON;

CONST MAX_SIZE = 127;

TYPE  POS_INTEGER        = 0 .. 32767;  {limited by the word-
                                         size of the computer}
      LIST_PTR           = ^LIST_ELEMENT;
      LIST_ELEMENT       = PACKED RECORD
                             HEAD, TAIL          : POS_INTEGER;
                             HEAD_SMALL          : BOOLEAN;
                             SUP_WEIGHT,
                             COST,
                             BASE_PRODUCT,
                             SIDE_PRODUCT        : INTEGER;
                             DESCENDANT, NEXT    : LIST_PTR
                           END;


VAR   W, CP      : ARRAY [1..MAX_SIZE] OF INTEGER;
      LIST, LEAF: LIST_PTR;
      N          : POS_INTEGER;


SEGMENT PROCEDURE INITIALIZING;
(**********************************************************)
(* Handles the inputs and initializing all the global    *)
(*   variables.                                          *)
(**********************************************************)
VAR I : INTEGER;

BEGIN
  WRITELN ('a linear algorithm to find all the h-arcs in',
                                         'the optimum');
  WRITELN ('   partition of a monotone basic polygon',
                                         '  (7/2/80)');
  WRITELN;

  {obtain the inputs}
  WRITE ('Please enter the size of the polygon (between 3',
                              ' and ',MAX_SIZE-1,'): ');
  READLN (N);
  WRITELN;
  WRITELN ('Now, starting from the smallest vertex and in',
                                                ' the ');
  WRITELN ('  clockwise direction, enter the weights of',
                                         ' the vertices:');
  FOR I := 1 TO N DO READ (W[I]);
  READLN;
  WRITELN;
```

```
  {calculate the cumulative adjacent
                                    products around the polygon}
  CP[1] := 0;
  FOR I := 2 TO N DO CP[I] := CP[I-1] + W[I-1] * W[I];

  {initialize the psuedo arc}
  NEW (LEAF);
  WITH LEAF^ DO
    BEGIN
      BASE_PRODUCT := 0;
      SIDE_PRODUCT := 0;
    END;

  {set up the output headings}
  WRITELN ('the potential h-arcs in the partitions are : ');

END; {initializing}



SEGMENT PROCEDURE ONE_SWEEP (VAR L : LIST_PTR);
(***********************************************************)
(* Sweep the polygon once, collects all potential h-arcs, *)
(*  puts them in a list.  The address of the head of the  *)
(*  list is stored in L.                                   *)
(***********************************************************)
VAR STACK                            : ARRAY [1..MAX_SIZE] OF
                                                 POS_INTEGER;
    TOP_ELEMENT, SECOND_ELEMENT,
    CURRENT, TOS                     : POS_INTEGER;
    P, ARC_LIST                      : LIST_PTR;


  PROCEDURE PUSH (C : INTEGER);
  (*********************************************************)
  (* Pushes the index C onto the stack and updates the    *)
  (*  variables TOS, TOP_ELEMENT, and SECOND_ELEMENT.     *)
  (*********************************************************)
  BEGIN
    STACK[TOS] := C;
    SECOND_ELEMENT := TOP_ELEMENT;
    TOP_ELEMENT := C;
    TOS := TOS - 1;
  END; {push}
```

```
  PROCEDURE POP_STACK;
  (*************************************************************)
  (* Pops the top element off the stack and updates the     *)
  (*  variables TOS, TOP_ELEMENT, and SECOND_ELEMENT.       *)
  (*************************************************************)
  BEGIN
    TOS := TOS + 1;
    TOP_ELEMENT := SECOND_ELEMENT;
    SECOND_ELEMENT := STACK[TOS + 2];
  END; {pop_stack}


(***************************************************************)
(* One-sweep begins here.                                     *)
(***************************************************************)
BEGIN
  {initialize the local variables}
  TOP_ELEMENT := 0;
  SECOND_ELEMENT := 0;
  STACK[N+1] := 0;
  TOS := N;
  ARC_LIST := NIL;
  PUSH (1);
  PUSH (2);
  CURRENT := 3;

  {scan through the polygon in the clockwise direction}
  WHILE CURRENT < N DO
    IF (W[SECOND_ELEMENT] <= W[TOP_ELEMENT]) AND
                                  (W[TOP_ELEMENT] > W[CURRENT])
    THEN
      BEGIN
        NEW(P);
        WITH P^ DO
          BEGIN
            HEAD := SECOND_ELEMENT;
            TAIL := CURRENT;
            HEAD_SMALL := W[HEAD] <= W[TAIL];
            BASE_PRODUCT := W[HEAD] * W[TAIL];
            SIDE_PRODUCT := CP[TAIL] - CP[HEAD];
            DESCENDANT := NIL;
            NEXT := ARC_LIST;
          END;
        ARC_LIST := P;

        POP_STACK;
        IF TOS >= (N-1) {there are less than 2
                                       elements on the stack}
```

```
            THEN
              BEGIN
                PUSH (CURRENT);
                CURRENT := CURRENT + 1;
              END;
          END
        ELSE
          BEGIN
            PUSH (CURRENT);
            CURRENT := CURRENT + 1;
          END;

      WHILE (TOS <= (N-3))
                  AND (W[SECOND_ELEMENT] <= W[TOP_ELEMENT])
                                  AND (W[TOP_ELEMENT] > W[N]) DO
        BEGIN
          NEW(P);
          WITH P^ DO
            BEGIN
              HEAD := SECOND_ELEMENT;
              TAIL := N;
              HEAD_SMALL := W[HEAD] <= W[TAIL];
              BASE_PRODUCT := W[HEAD] * W[TAIL];
              SIDE_PRODUCT := CP[TAIL] - CP[HEAD];
              DESCENDANT := NIL;
              NEXT := ARC_LIST;
            END;
          ARC_LIST := P;
          POP_STACK;
        END;

      L := ARC_LIST;
    END; {one_sweep}
```

```
SEGMENT PROCEDURE MONO_PARTITION (VAR L : LIST_PTR);
(************************************************************)
(* Obtains all the h-arcs in the optimum partition of the  *)
(*  polygon and returns them in a list.  The address of    *)
(*  the head of the list is stored in L.                   *)
(************************************************************)

  FUNCTION FAN_COST (HR, HC : LIST_PTR) : INTEGER;
  (*********************************************************)
  (* Calculates the cost of the fan of the subpolygon     *)
  (*  bounded above by HC and below by HR.                *)
  (*********************************************************)
  VAR TEMP1, TEMP2 : INTEGER;

  BEGIN
    TEMP1 := HR^.SIDE_PRODUCT - HC^.SIDE_PRODUCT
                                    + HC^.BASE_PRODUCT;
    WITH HR^ DO
      IF HEAD_SMALL
      THEN
        BEGIN
          IF HEAD = HC^.HEAD
          THEN TEMP2 := HC^.BASE_PRODUCT
          ELSE TEMP2 := CP[HEAD+1] - CP[HEAD];
          FAN_COST := (TEMP1 - TEMP2) * W[HEAD];
        END
      ELSE
        BEGIN
          IF TAIL = HC^.TAIL
          THEN TEMP2 := HC^.BASE_PRODUCT
          ELSE TEMP2 := CP[TAIL] - CP[TAIL-1];
          FAN_COST := (TEMP1 - TEMP2) * W[TAIL];
        END;
  END; {fan_cost}


  FUNCTION SUPPORTING_WEIGNT (HR, HC : LIST_PTR) : INTEGER;
  (*********************************************************)
  (* Find the supporting weight of the subpolygon bounded *)
  (*  above by HC and below by HR.                         *)
  (*********************************************************)
  VAR Y : INTEGER;

  BEGIN
    {calculate the denominator}
    Y := (HR^.SIDE_PRODUCT - HR^.BASE_PRODUCT)
                 - (HC^.SIDE_PRODUCT - HC^.BASE_PRODUCT);

    {calculate the SUPPORTING_WEIGHT}
    SUPPORTING_WEIGHT := (HR^.COST + Y - 1) DIV Y;
                                         {ceiling function}
  END; {supporting_weight}
```

```
PROCEDURE REMOVE (VAR S : LIST_PTR; MIN : INTEGER);
(**************************************************************)
(* Removes all the arcs in S whose SUP_WEIGHTS are equal to *)
(*or larger than MIN from the list.                         *)
(**************************************************************)
VAR NOT_DONE : BOOLEAN;

BEGIN
  NOT_DONE := TRUE;

  WHILE NOT_DONE DO
      IF S = NIL
      THEN NOT_DONE := FALSE
      ELSE
        IF S^.SUP_WEIGHT < MIN
        THEN NOT_DONE := FALSE
        ELSE S := S^.NEXT;
END; {remove}


PROCEDURE SUB_PARTITION (VAR S : LIST_PTR; MIN : INTEGER);
(**************************************************************)
(* Finds the optimum partition of the subpolygonbounded *)
(*  below by the potential h-arc at the head of S.  The *)
(*  h-arcs in the optimum partition of the subpolygon   *)
(*  is kept in a list with S pointing to the head of    *)
(*  the list.                                           *)
(**************************************************************)
VAR TEMP     : INTEGER;
    TEMP_PTR : LIST_PTR;
    NOT_DONE : BOOLEAN;

BEGIN
  IF S^.NEXT <> NIL
  THEN
    BEGIN
      IF S^.HEAD_SMALL
      THEN TEMP := W[S^.HEAD]
      ELSE TEMP := W[S^.TAIL];
      SUB_PARTITION (S^.NEXT,TEMP);     {S^.NEXT may become
                                         NIL when return
                                         from SUB_PARTITON}
    END;

  IF S^.NEXT = NIL
  THEN TEMP_PTR := LEAF  {S is the last arc in the list}
                         {LEAF is a psuedo arc with
                          both LEAF^.BASE_PRODUCT and
                          LEAF^.SIDE_PRODUCT equal to NIL}
  ELSE TEMP_PTR := S^.NEXT;
```

```
    S^.COST := FAN_COST(S,TEMP_PTR);
    NOT_DONE := TRUE;
    WHILE NOT_DONE DO
      BEGIN
        S^.SUP_WEIGHT := SUPPORTING_WEIGHT(S,TEMP_PTR);
        IF S^.SUP_WEIGHT > = MIN  {to see if the partition is
                                   optimum in the subpolygon}

        THEN
          BEGIN
            REMOVE (S,MIN);     {delete all h-arcs not in the
                                 optimum partition of the
                                 subpolygon}
            NOT_DONE := FALSE;
          END
        ELSE
          BEGIN
            IF S^.NEXT <> NIL
            THEN
              IF S^.NEXT^.SUP_WEIGHT <= S^.SUP_WEIGHT
              THEN NOT_DONE := FALSE
              ELSE
                BEGIN        {condense S^.NEXT into S}
                  TEMP_PTR := S^.NEXT;
                  S^.NEXT := TEMP_PTR^.NEXT;
                  S^.COST := S^.COST + TEMP_PTR^.COST;
                  TEMP_PTR^.NEXT := S^.DESCENDANT;
                  S^.DESCENDANT := TEMP_PTR;
                  IF S^.NEXT = NIL
                  THEN TEMP_PTR := LEAF
                  ELSE TEMP_PTR := S^.NEXT;
                END
            ELSE NOT_DONE := FALSE;
          END;
      END;
  END; {sub_partition}


BEGIN
  SUB_PARTITION (L,W[1]);
END; {mono_partition}
```

```
PROCEDURE WRITE_LIST (L : LIST_PTR; MIN, INDENT : INTEGER);
(*****************************************************************)
(* Displays the h-arcs in the list pointed by L.                 *)
(*****************************************************************)
VAR TEMP : POS_INTEGER;

BEGIN
  WHILE L <> NIL DO
    BEGIN
      IF L^.HEAD_SMALL
      THEN TEMP := L^.HEAD
      ELSE TEMP := L^.TAIL;
      IF TEMP <> MIN
      THEN WRITELN (' ':INDENT,L^.HEAD,' ':3,L^.TAIL);
      WRITE_LIST (L^.DESCENDANT,TEMP,INDENT+3);
      L := L^.NEXT;
    END;
END; {write_list}


BEGIN {main program begins here}
  INITIALIZING;
  ONE_SWEEP (LIST);
  MONO_PARTITION (LIST);
  IF LIST <> NIL
  THEN WRITE_LIST (LIST,1,3)
  ELSE WRITELN (' ':3,'NIL');
END. {main program}
```

Appendix II

```
PROGRAM OPTIMUM_PARTITION_OF_A_GENERAL_CONVEX_POLYGON;

CONST MAX_SIZE = 127;    {the maximum number of vertices in
                                              a polygon is 126}
      MAX_INT  = 32767; {the largest integer in the machine}

TYPE  POS_INTEGER         = 0 .. MAX_INT;
      LIST_PTR            = ^LIST_ELEMENT;
      LIST_ELEMENT        = PACKED RECORD
                              HEAD          : POS_INTEGER;
                              STAY          : BOOLEAN;
                              TAIL          : POS_INTEGER;
                              HEAD_SMALL    : BOOLEAN;
                              NEXT          : LIST_PTR
                            END;
      TREE_PTR            = ^TREE_ELEMENT;
      TREE_ELEMENT        = PACKED RECORD
                              HEAD, TAIL              : POS_INTEGER;
                              HEAD_SMALL              : BOOLEAN;
                              SUP_WEIGHT,
                              TREE_COST,
                              TREE_BASE_PRODUCT,
                              TREE_SIDE_PRODUCT,
                              LOCAL_COST,
                              LOCAL_BASE_PRODUCT,
                              LOCAL_SIDE_PRODUCT: INTEGER;
                              DESCENDANT, NEXT,
                              H_ARC, V_ARC            : TREE_PTR;
                              LIST_LINK               : LIST_PTR;
                              DEPTH                   : INTEGER
                            END;

                            {V_ARC and H_ARC are used in two
                             different ways : (1) they are
                             used to link the unprocessed arcs
                             together to form an arc-tree; and
                             (2) they are used as the left
                             link and the right link of the
                             processed arcs in the leftist
                             tree for the priority queue.     }


VAR   W, CP               : ARRAY [1..MAX_SIZE] OF INTEGER;
      LIST1, LIST2        : LIST_PTR;
      V_TREE, H_TREE      : TREE_PTR;
      N                   : POS_INTEGER;
```

```
SEGMENT PROCEDURE INITIALIZING;
(**************************************************************)
(* Handles the inputs and initializing all the global        *)
(*  variables.                                                *)
(**************************************************************)
VAR I : INTEGER;

BEGIN
  WRITELN ('a linear algorithm to find all the h-arcs in');
  WRITELN ('   the optimum partition of a convex polygon',
                                                 '   (7/16/80)');
  WRITELN;

  {obtain the inputs}
  WRITE ('Please enter the size of the polygon (between 3',
                                   ' and ',MAX_SIZE-1,'): ');
  READLN (N);
  WRITELN;
  WRITELN ('Now, starting from the smallest',
                                      ' vertex and in the ');
  WRITELN ('  clockwise direction, ',
                        'enter the weights of the vertices:');
  FOR I := 1 TO N DO READ (W[I]);
  READLN;
  WRITELN;

  {calculate the cumulative adjacent
                                  products around the polygon}
  CP[1] := 0;
  FOR I := 2 TO N DO CP[I] := CP[I-1] + W[I-1] * W[I];

  {set up the output headings}
  WRITELN ('the potential h-arcs in the partitions are : ');

END; {initializing}
```

```
SEGMENT PROCEDURE ONE_SWEEP (VAR L : LIST_PTR);
(*************************************************************)
(* Sweep the polygon once, collects all potential h-arcs, *)
(*  puts them in a list.  The address of the head of the  *)
(*  list is stored in L.                                   *)
(*************************************************************)
VAR STACK                              : ARRAY [1..MAX_SIZE] OF
                                                 POS_INTEGER;
    TOP_ELEMENT, SECOND_ELEMENT,
    CURRENT, TOS                       : POS_INTEGER;
    P, ARC_LIST                        : LIST_PTR;


  PROCEDURE PUSH (C : INTEGER);
  (*********************************************************)
  (* Pushes the index C onto the stack and updates the    *)
  (*  variables TOS, TOP_ELEMENT, and SECOND_ELEMENT.     *)
  (*********************************************************)
  BEGIN
    STACK[TOS] := C;
    SECOND_ELEMENT := TOP_ELEMENT;
    TOP_ELEMENT := C;
    TOS := TOS - 1;
  END; {push}


  PROCEDURE POP_STACK;
  (*********************************************************)
  (* Pops the top element off the stack and updates the   *)
  (*  variables TOS, TOP_ELEMENT, and SECOND_ELEMENT.     *)
  (*********************************************************)
  BEGIN
    TOS := TOS + 1;
    TOP_ELEMENT := SECOND_ELEMENT;
    SECOND_ELEMENT := STACK[TOS + 2];
  END; {pop_stack}


BEGIN {one_sweep begins here}
  {initialize the local variables}
  TOP_ELEMENT := 0;
  SECOND_ELEMENT := 0;
  STACK[N+1] := 0;
  TOS := N;
  ARC_LIST := NIL;
  PUSH (1);
  PUSH (2);
  CURRENT := 3;
```

```
{scan through the polygon in the clockwise direction}
WHILE CURRENT < N DO
  IF (W[SECOND_ELEMENT] <= W[TOP_ELEMENT]) AND
                                   (W[TOP_ELEMENT] > W[CURRENT])
  THEN
    BEGIN
      NEW(P);
      WITH P^ DO
        BEGIN
          HEAD := SECOND_ELEMENT;
          TAIL := CURRENT;
          STAY := FALSE;
          HEAD_SMALL := W[HEAD] <= W[TAIL];
          NEXT := ARC_LIST;
        END;
      ARC_LIST := P;

      POP_STACK;
      IF TOS >= (N-1) {there are less than
                                   2 elements on the stack}
      THEN
        BEGIN
          PUSH (CURRENT);
          CURRENT := CURRENT + 1;
        END;
    END
  ELSE
    BEGIN
      PUSH (CURRENT);
      CURRENT := CURRENT + 1;
    END;

WHILE (TOS <= (N-3))
         AND (W[SECOND_ELEMENT] <= W[TOP_ELEMENT])
                                AND (W[TOP_ELEMENT] > W[N]) DO
  BEGIN
    NEW(P);
    WITH P^ DO
      BEGIN
        HEAD := SECOND_ELEMENT;
        TAIL := N;
        STAY := FALSE;
        HEAD_SMALL := W[HEAD] <= W[TAIL];
        NEXT := ARC_LIST;
      END;
    ARC_LIST := P;
    POP_STACK;
  END;

L := ARC_LIST;
END; {one_sweep}
```

```
SEGMENT PROCEDURE BUILD_TREE (VAR L : LIST_PTR;
     VAR VT, HT : TREE_PTR; FIRST, LAST, MIN : POS_INTEGER);
(*****************************************************************)
(* Traces all the arcs in the list pointed by L and             *)
(*  build an arc-tree with the root pointed by T.               *)
(*****************************************************************)
VAR NOT_DONE     : BOOLEAN;
    P            : TREE_PTR;
    Q            : LIST_PTR;

BEGIN
  NOT_DONE := TRUE;
  VT := NIL;
  HT := NIL;
  WHILE NOT_DONE DO
    IF L = NIL
    THEN NOT_DONE := FALSE
    ELSE
      IF (L^.HEAD < FIRST) OR (L^.TAIL > LAST)
      THEN NOT_DONE := FALSE
      ELSE
        BEGIN
          Q := L^.NEXT;
          IF L^.HEAD <> 1
          THEN
            BEGIN
              NEW (P);
              WITH P^ DO
                BEGIN
                  HEAD := L^.HEAD;
                  TAIL := L^.TAIL;
                  HEAD_SMALL := L^.HEAD_SMALL;
                  DESCENDANT := NIL;
                  DEPTH := 1;
                  LIST_LINK := L;
                  {LOCAL_COST, LOCAL_BASE_PRODUCT,
                   LOCAL_SIDE_PRODUCT, TREE_COST,
                   TREE_BASE_PRODUCT, TREE_SIDE_PRODUCT,
                   H_ARC, and V_ARC are undefined at this
                   point                                         }
                  IF (HEAD_SMALL AND (HEAD = MIN)) OR
                           (NOT HEAD_SMALL AND (TAIL = MIN))
                  THEN
                    BEGIN
                      NEXT := VT;
                      VT := P;
                    END
                  ELSE
                    BEGIN
                      NEXT := HT;
                      HT := P;
                    END;
```

```
                    IF HEAD_SMALL
                    THEN BUILD_TREE (Q,V_ARC,H_ARC,
                                        HEAD,TAIL,HEAD)
                    ELSE BUILD_TREE (Q,V_ARC,H_ARC,
                                        HEAD,TAIL,TAIL);
                    {note that there will be at most one arc
                     in the V_ARC list but may be several arcs
                     in the H_ARC list                       }
                 END;
              END;
            L := Q;
          END;
END; {build_tree}


SEGMENT PROCEDURE POLY_PARTITION (VAR T : TREE_PTR);
(************************************************************)
(* To find all the h-arcs that are present in the optimum *)
(*  partition of the polygon and returns them in the arc- *)
(*  tree pointed by T.                                     *)
(************************************************************)


  PROCEDURE FAN_COST (T : TREE_PTR);
  (**********************************************************)
  (* To find the cost of the fan of the subpolygon bounded*)
  (*  below by the arc pointed by T and above by the arcs *)
  (*  pointed by T^.H_ARCs and T^.V_arcs.                 *)
  (**********************************************************)
  VAR X : POS_INTEGER;
      Y, S1, S2 : INTEGER;

  BEGIN
    WITH T^ DO
      BEGIN
        IF HEAD_SMALL
        THEN
          BEGIN
            IF V_ARC = NIL
            THEN
              BEGIN
                X := HEAD + 1;
                S1 := CP[X] - CP[HEAD];
              END
            ELSE
              BEGIN
                X := V_ARC^.TAIL;
                S1 := V_ARC^.TREE_BASE_PRODUCT;
              END;
```

```
          S2 := (CP[TAIL] - CP[X]);
          Y := W[HEAD];
        END
      ELSE
        BEGIN
          IF V_ARC = NIL
          THEN
            BEGIN
              X := TAIL - 1;
              S1 := CP[TAIL] - CP[X];
            END
          ELSE
            BEGIN
              X := V_ARC^.HEAD;
              S1 := V_ARC^.TREE_BASE_PRODUCT;
            END;
          S2 := (CP[X] - CP[HEAD]);
          Y := W[TAIL];
        END;

      IF H_ARC <> NIL
      THEN S2 := S2 - H_ARC^.TREE_SIDE_PRODUCT
                      + H_ARC^.TREE_BASE_PRODUCT;
      {all the SIDE_PRODUCTs and the BASE_PRODUCTs are
       added together and stored in the root of the
       leftist tree pointed by H_ARC                    }

      LOCAL_COST := S2 * Y;
      LOCAL_SIDE_PRODUCT := S1 + S2;
      LOCAL_BASE_PRODUCT := W[HEAD] * W[TAIL];
    END;
END; {fan_cost}


PROCEDURE SUPPORTING_WEIGHT (T : TREE_PTR);
(**********************************************************)
(* To find the supporting weight of the arc pointed by T*)
(*  with respect to the subpolygon bounded below by the *)
(*  arc pointed by T and above by the arcs pointed by   *)
(*  the T^.H_ARC and T^.V_ARC.                           *)
(**********************************************************)
VAR D : INTEGER;

BEGIN
  WITH T^ DO
    BEGIN
      D := (LOCAL_SIDE_PRODUCT - LOCAL_BASE_PRODUCT);
      SUP_WEIGHT := (LOCAL_COST + D - 1) DIV D;
                                        {ceiling function}
    END;
END; {supporting_weight}
```

```
FUNCTION MERGE (T1, T2 : TREE_PTR) : TREE_PTR;
(*************************************************************)
(* Merges two leftist trees into one and returns it in   *)
(*  MERGE.                                                   *)
(*************************************************************)
VAR TEMP_PTR                          : TREE_PTR;
    TEMP_COST, TEMP_BASE_PRODUCT,
    TEMP_SIDE_PRODUCT                 : INTEGER;

BEGIN
  IF T2 = NIL
  THEN MERGE := T1
  ELSE
    IF T1 = NIL
    THEN MERGE := T2
    ELSE
      BEGIN
        TEMP_COST := T1^.TREE_COST + T2^.TREE_COST;
        TEMP_SIDE_PRODUCT := T1^.TREE_SIDE_PRODUCT
                                 + T2^.TREE_SIDE_PRODUCT;
        TEMP_BASE_PRODUCT := T1^.TREE_BASE_PRODUCT
                                 + T2^.TREE_BASE_PRODUCT;
        IF T1^.SUP_WEIGHT < T2^.SUP_WEIGHT
        THEN
          BEGIN
            TEMP_PTR := T1;
            T1 := T2;
            T2 := TEMP_PTR;
          END;

        WITH T1^ DO
          BEGIN
            H_ARC := MERGE (H_ARC,T2);
            {H_ARC never equals NIL at this point}
            IF V_ARC = NIL
            THEN
              BEGIN
                V_ARC := H_ARC;
                H_ARC := NIL
              END
            ELSE
              BEGIN
                IF V_ARC^.DEPTH < H_ARC^.DEPTH
                THEN
                  BEGIN
                    TEMP_PTR := V_ARC;
                    V_ARC := H_ARC;
                    H_ARC := TEMP_PTR;
                  END;
                DEPTH := H_ARC^.DEPTH + 1;
              END;
            TREE_COST := TEMP_COST;
```

```
            TREE_SIDE_PRODUCT := TEMP_SIDE_PRODUCT;
            TREE_BASE_PRODUCT := TEMP_BASE_PRODUCT;
          END;
        MERGE := T1;
      END;
END; {merge}


FUNCTION CONDENSE (T : TREE_PTR; MIN : INTEGER) : BOOLEAN;
(**********************************************************)
(* CONDENSE is set to false if T = NIL or T^.SUP_WEIGHT *)
(*  <= MIN                                              *)
(**********************************************************)
BEGIN
  IF T = NIL
  THEN CONDENSE := FALSE
  ELSE CONDENSE := T^.SUP_WEIGHT > MIN;
END; {condense}


PROCEDURE COMBINE (VAR T : TREE_PTR; V_FLAG : BOOLEAN);
(**********************************************************)
(* If V_FLAG, it combines the arc pointed by T^.V_ARC    *)
(*  into the arc pointed by T, else it combines the arc *)
(*  pointed by T^.H_ARC into the arc pointed by T.  In  *)
(*  either case, the arc to be combined is deleted from *)
(*  the corresponding leftist tree and put into the     *)
(*  DESCENDANT list of the parent.                      *)
(**********************************************************)
VAR TEMP_PTR : TREE_PTR;

BEGIN
  IF V_FLAG
  THEN
    BEGIN
      TEMP_PTR := T^.V_ARC;
      T^.V_ARC := MERGE (TEMP_PTR^.V_ARC,TEMP_PTR^.H_ARC);
    END
  ELSE
    BEGIN
      TEMP_PTR := T^.H_ARC;
      T^.H_ARC := MERGE (TEMP_PTR^.V_ARC,TEMP_PTR^.H_ARC);
    END;
  TEMP_PTR^.V_ARC := NIL;
  TEMP_PTR^.H_ARC := NIL;
  TEMP_PTR^.NEXT := T^.DESCENDANT;
  T^.DESCENDANT := TEMP_PTR;
  T^.LOCAL_COST := T^.LOCAL_COST + TEMP_PTR^.LOCAL_COST;
  T^.LOCAL_SIDE_PRODUCT := T^.LOCAL_SIDE_PRODUCT
              + TEMP_PTR^.LOCAL_SIDE_PRODUCT
                                - TEMP_PTR^.LOCAL_BASE_PRODUCT;
END; {combine}
```

```
PROCEDURE REMOVE (VAR T : TREE_PTR; MIN : INTEGER);
(*************************************************************)
(* Removes all the arcs in the leftist tree pointed by T*)
(*   whose SUP_WEIGHTs are larger than or equal to MIN.      *)
(*************************************************************)
VAR NOT_DONE : BOOLEAN;

BEGIN
  NOT_DONE := TRUE;
  WHILE NOT_DONE DO
    IF T = NIL
    THEN NOT_DONE := FALSE
    ELSE
      IF T^.SUP_WEIGHT < MIN
      THEN NOT_DONE := FALSE
      ELSE T := MERGE (T^.V_ARC,T^.H_ARC);
END; {remove}


PROCEDURE SUB_PARTITION (VAR TREE : TREE_PTR;
                                            MIN : INTEGER);
(*************************************************************)
(* To find the optimum partition of the subpolygon          *)
(*  bounded below by the root of the arc-tree pointed        *)
(*  by T.                                                     *)
(*************************************************************)
VAR T, R, P, TEMP_PTR : TREE_PTR;
    TEMP              : INTEGER;
    NOT_DONE, FLAG    : BOOLEAN;

BEGIN
  T := TREE;
  R := NIL;
  WHILE T <> NIL DO
    BEGIN
      P := T^.NEXT;
      T^.NEXT := NIL;
      IF T^.HEAD_SMALL
      THEN TEMP := W[T^.HEAD]
      ELSE TEMP := W[T^.TAIL];
      IF T^.H_ARC <> NIL
      THEN SUB_PARTITION (T^.H_ARC,TEMP);
          {when return, all the h-arcs in the subpolygon
           will be put in a priority queue                  }

      IF T^.V_ARC <> NIL
      THEN SUB_PARTITION (T^.V_ARC,TEMP);
          {there should be at most 1 v-arc, i.e.
           T^.V_ARC^.NEXT = NIL, when return, all the
           h-arcs in the subpolygon will be put in a
           priority queue                                    }
```

```
{calculate the cost of the fan of the subpolygon
 bounded below by the arc pointed by T and above by
 the v-arcs and h-arcs of T                          }
FAN_COST(T);

NOT_DONE := TRUE;
FLAG := TRUE;
WHILE NOT_DONE DO
  BEGIN
    {calculate the supporting weight
                              of the arc pointed by T}
    SUPPORTING_WEIGHT (T);

    IF T^.SUP_WEIGHT >=MIN {to see if the partition
                            is optimum in the
                            subpolygon              }
    THEN
      BEGIN
       REMOVE (T,MIN);     {delete all h-arcs not
                            in the optimum partition
                            of the subpolygon       }
        NOT_DONE := FALSE;
        FLAG := FALSE;
      END
    ELSE
      IF CONDENSE (T^.V_ARC,T^.SUP_WEIGHT)
      THEN COMBINE (T, TRUE)
      ELSE
        IF CONDENSE (T^.H_ARC,T^.SUP_WEIGHT)
        THEN COMBINE (T, FALSE)
        ELSE NOT_DONE := FALSE;
  END;

{maintain the leftist tree structure}
IF FLAG
THEN
  BEGIN
    T^.TREE_COST := T^.LOCAL_COST;
    T^.TREE_SIDE_PRODUCT := T^.LOCAL_SIDE_PRODUCT;
    T^.TREE_BASE_PRODUCT := T^.LOCAL_BASE_PRODUCT;
    IF T^.V_ARC <> NIL
    THEN
      BEGIN
        T^.TREE_COST := T^.TREE_COST
                             + T^.V_ARC^.TREE_COST;
        T^.TREE_SIDE_PRODUCT := T^.TREE_SIDE_PRODUCT
                  + T^.V_ARC^.TREE_SIDE_PRODUCT
                        - T^.V_ARC^.TREE_BASE_PRODUCT;
      END;
```

```
                IF T^.H_ARC <> NIL
                THEN
                  BEGIN
                    T^.TREE_COST := T^.TREE_COST
                                      + T^.H_ARC^.TREE_COST;
                    T^.TREE_SIDE_PRODUCT := T^.TREE_SIDE_PRODUCT
                           + T^.H_ARC^.TREE_SIDE_PRODUCT
                                 - T^.H_ARC^.TREE_BASE_PRODUCT;
                  END;

                IF T^.V_ARC <> NIL
                THEN
                  IF T^.H_ARC <> NIL
                  THEN
                    BEGIN
                      IF T^.V_ARC^.DEPTH < T^.H_ARC^.DEPTH
                      THEN
                        BEGIN
                          TEMP_PTR := T^.V_ARC;
                          T^.V_ARC := T^.H_ARC;
                          T^.H_ARC := TEMP_PTR;
                        END;
                      T^.DEPTH := T^.H_ARC^.DEPTH + 1;
                    END
                  ELSE
                ELSE
                  IF T^.H_ARC <> NIL
                  THEN
                    BEGIN
                      T^.V_ARC := T^.H_ARC;
                      T^.H_ARC := NIL;
                    END;
              END;

            R := MERGE (R,T);
            T := P;
          END;
        TREE := R;
      END; {sub_partition}


  BEGIN {poly_partition begins here}
    SUB_PARTITION (T,W[1]);
  END; {poly_partition}
```

```
PROCEDURE MARK_LIST (T : TREE_PTR);
(*************************************************************)
(* Traverses the tree pointed by T preorderly, finds out   *)
(*   all the potential h-arcs which are present in the     *)
(*   optimum partition of the polygon and marks the        *)
(*   corresponding elements in the list pointed by LIST1.  *)
(*************************************************************)
BEGIN
  WHILE T <> NIL DO
    BEGIN
      T^.LIST_LINK^.STAY := TRUE;
      MARK_LIST (T^.DESCENDANT);
      MARK_LIST (T^.V_ARC);
      MARK_LIST (T^.H_ARC);
      T := T^.NEXT;
    END;
END; {mark_list}



PROCEDURE WRITE_LIST (VAR L : LIST_PTR;
                          FIRST, LAST, MIN, INDENT : INTEGER);
(*************************************************************)
(* Displays the h-arcs in the list pointed by L.           *)
(*************************************************************)
VAR TEMP          : POS_INTEGER;
    NOT_DONE      : BOOLEAN;

BEGIN
  NOT_DONE := TRUE;
  WHILE NOT_DONE DO
    IF L = NIL
    THEN NOT_DONE := FALSE
    ELSE
      IF (L^.HEAD < FIRST) OR (L^.TAIL > LAST)
      THEN NOT_DONE := FALSE
      ELSE
        BEGIN
          IF L^.STAY
          THEN
            BEGIN
              IF L^.HEAD_SMALL
              THEN TEMP := L^.HEAD
              ELSE TEMP := L^.TAIL;
              IF TEMP <> MIN
              THEN
                BEGIN
                  WRITELN (' ':INDENT,
                                    L^.HEAD,' ':3,L^.TAIL);
```

```
                    WRITE_LIST (L^.NEXT,L^.HEAD,
                                         L^.TAIL,TEMP,INDENT+3);
                  END;
              END;
            L := L^.NEXT;
          END;
END; {write_list}


(**************************************************************)
(* main program begins here.                                  *)
(**************************************************************)
BEGIN
  INITIALIZING;
  ONE_SWEEP (LIST1);
  LIST2 := LIST1;
  BUILD_TREE (LIST2,V_TREE,H_TREE,1,N,1);      {V_TREE = NIL}
  POLY_PARTITION (H_TREE);
  IF H_TREE = NIL
  THEN WRITELN (' ':3,'NIL')
  ELSE
    BEGIN
      MARK_LIST (H_TREE);
      WRITE_LIST (LIST1,1,N,1,3);
    END;
END. {main program}
```

DATE

ILME